UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE GRAU

TFG TITLE: A diagnosis and prevention system for mosquito transmitted diseases

DEGREE: Bachelor's degree in Air Navigation Engineering and Bachelor's Degree in Telecommunications Systems Engineering

AUTHOR: Anna Reig Callis

ADVISOR: Jaime Oscar Casas Piedrafita

DATE: July 7, 2019

Títol: Un sistema de diagnòstic i prevenció de malalties transmeses per mosquits
Autor: Anna Reig Callis

Director: Jaime Oscar Casas Piedrafita

Data: 7 de juliol de 2019

Resum

L'objectiu principal d'aquest treball és dissenyar un sistema integral capaç de diagnosticar i prevenir la malària, una malaltia transmesa per una senzilla picada de mosquit. Així doncs, el sistema dissenyat consta de dues vessants: el diagnòstic, que ha de ser portàtil, fàcil d'utilitzar, ràpid, econòmic i accessible a tots els habitants del món, i la prevenció, que ha de ser ràpida, eficaç, localitzada, econòmica i respectuosa amb ecosistema.
Pel que fa al diagnòstic, el seu objectiu principal és controlar qualsevol persona ubicada a qualsevol lloc del món des de qualsevol hospital del món. Mentre es generen les dades d'un pacient, i fins i tot després de la seva generació, aquestes poden ser reproduïdes i avaluades per qualsevol professional mèdic des de qualsevol hospital. Per tant, sense importar on estigui físicament el pacient, aquest pot ser monitoritzat per qualsevol metge. El sistema prototipat consta d'un hardware, que genera dades de les constants vitals principals del pacient, un back-end, que les captura, un front-end, que les mostra i les recupera i, una base de dades, que les salva. Una vegada el pacient sap com funciona el sistema, no requereix de cap assistència mèdica per prendre mesures. A més, el sistema es caracteritza per ser petit i lleuger, el que permet transportar-lo fàcilment a qualsevol lloc. També s'ha de destacar la rapidesa amb la que el sistema pren i guarda mesures per tal de que estiguin disponibles per a la seva avaluació mèdica.
Pel que fa a la prevenció, el seu principal objectiu és determinar la ubicació de zones que allotgin mosquits i les seves larves. Un cop situades, es tracta només d'alertar la població més propera del risc existent a la zona, alliberant mosquits masculins estèrils, en cas de mosquits o aplicant el corresponent mètode d'extermini, en cas de larves. El que s'aconsegueix mitjançant la detecció abans de prendre qualsevol acció, és avaluar la necessitat real de l'acció i ser més eficaç duent-la a terme.
D'aquesta manera, fotografiant una àrea determinada, el sistema prototipat és capaç de localitzar i identificar aigües estancades i de localitzar i classificar mosquits. La detecció es realitza sempre amb l'ajut de sensors de terra i aire. Una càmera web instal·lada en un drone, en un vehicle mòbil o en un lloc estàtic, s'encarrega de fer les fotografies. Així, en cas de no tenir una càmera estàtica, després de l'exploració de l'àrea d'interès, s'obté un mapa que permet estimar la densitat de mosquits de la zona. L'avantatge més important d'utilitzar aquest sistema és que, combinant la Intel·ligència Artificial per a la detecció de bassals i mosquits amb dades mediambientals donades pels sensors de terra i aire, la probabilitat de detectar amb èxit augmenta notablement.

Title : A diagnosis and prevention system for mosquito transmitted diseases
Author: Anna Reig Callis

Advisor: Jaime Oscar Casas Piedrafita

Date: July 7, 2019

Overview

The main objective of bachelor project is to design a comprehensive system able to diagnose and prevent the malaria, a tropical disease transmitted by a simple mosquito bite.
Therefore, this system designed has two different perspectives: the diagnostic which should be portable, easy-to-use, fast, cheap and available for all world's inhabitants, and the prevention which should be fast, effective, localized, cheap and respectful with the ecosystem.
Regarding the diagnostic approach, its main aim is to monitor any person located in any place of the world from any hospital of the world. While the data of a patient are being generated and even after their generation, they can be reproduced and evaluated by any medical professionals at any hospital. Thus, no matter where the patient is physically, thanks to the prototyped system, he or she can be controlled by any doctor.
In this way, the prototyped system is composed of a hardware which generates data of the patient principal vital constants, a back-end which captures them, a front-end which shows and retrieves them and a database which saves them. Once the patient knows how the system works, this easy-to-use system does not require any medical assistance to start taking measures. Moreover, it is small and lightweight which allows to carry it easily to anywhere. The rapidity into which the system takes measures and saves them to make them available for medical evaluation should be also highlighted.
Concerning the prevention approach, its main objective is to determine the location of zones accommodating mosquitoes and their larvae. Once these zones situated, it is only about: either alerting the closest population of the existent risk in the zone, releasing infertile male mosquitoes, in case of mosquitoes or, applying the corresponding extermination method, in case of larvae. What is achieved through the detection before taking any action, is to assess the real need of taking the action and to be more effective when the need is real respecting the ecosystem to the maximum.
In this way, from pictures taken in a certain area, the prototyped system is able to locate and identify stagnant waters and to locate and classify mosquitoes. The detection or identification is always carried out with the complementary help of ground and air sensors. A web camera fitted in a drone, in a rover or placed statically, is in charge of taking such pictures. Thus, in case of not having a static camera, after the studied area scanning, what it is finally achieved is a map, from which the mosquito density in the studied area can be known.
The most important benefit of using such system is that by combining Artificial Intelligence for detecting puddles and mosquitoes with environmental data given by ground and air sensors, the probability of successfully detecting increases a lot.
Above all, efforts have been made to design the cheapest possible systems so that it is available for the maximum number of people.

A l'ansietat,
per fer-me créixer i obrir-me els ulls
A la meva família, als meus pares i al meu germà,
perque sense ells no hagués arribat fins aquí

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI	Artificial Intelligence
aVR	augmented Vector Right
aVL	augmented Vector Left
aVF	augmented Vector Foot
BPM	Beats Per Minute
CNN	Convolutional Neural Network
CV	Coefficient of variation
DBP	Diastolic Blood Pressure
ECG	Electrocardiogram
HR	Heart Rate
HW	Hardware
MAP	Mean Arterial Pressure
MEE	Medical Electrical Equipment
MMP	Multi Parametric Module
NIBP	Non-Invasive Blood Pressure
NSSM	Non-Sucking Service Manager
OEM	Original Equipment Manufacturer
PI	Perfusion Index
PR	Pulse Rate
RESP	Respiration variation
RPM	Respirations Per Minute
SBP	Systolic Blood Pressure
SQL	Structured Query Language
ST	ST segment
SW	Software
TEMP	Temperature variation
UART	Universal Asynchronous Receiver-Transmitter
UI	User Interface

INTRODUCTION

In 2016, there were 216 million malaria cases that led to 440,000 deaths, from which about two thirds (290,000) were children under five years of age. This can be translated into a daily toll of nearly 800 children under age 5. All these facts make malaria the first cause of child deaths in Africa. Nevertheless, malaria does not only affects Africa but also Center America, South America, South Asia and it is emerging again in Europe after being eradicated some decades ago, due to the increase of temperatures.

Thus, malaria, a tropical disease transmitted by a simple mosquito bite, is an urgent public health priority at a global level. In fact, the disease itself and the costs of treatment trap families in a cycle of illness, suffering and poverty. Today, 3.2 billion (almost half of the world population) are at risk. Since 2000, malaria has cost sub-Saharan Africa US$ 300 million each year for case management alone in Africa.

For all the aforementioned reasons, this bachelor project is focus on the fight against malaria, a worldwide increasing problem due to the raise of the global temperature.

Therefore, the fight against malaria has been addressed from a comprehensive system, composed of two different perspectives: the diagnostic which should be portable, easy-to-use, fast, cheap and available for all world's inhabitants, and the prevention which should be fast, effective, localized, cheap and respectful with the ecosystem.

Regarding the diagnostic approach, it is based on the symptoms caused by malaria, which can be basically summarized to fever and headache. On one hand, fever provokes an increase in body temperature causing abnormalities in the electrocardiogram shape as well as in the heart rate, pulse rate and respiration rate of the affected patient. On the other hand, regarding headaches, when they are strong, they also originate noticeable alterations in the electrocardiogram.

Hence, the main aim of this approach is to monitor any person located in any place of the world from any hospital of the world. While the data of a patient are being generated and even after their generation, they can be reproduced and carefully evaluated by medical professionals at any hospital. Thus, no matter where the patient is, he or she can be controlled by any doctor thanks to the prototyped system, which implements the concept of telemedicine.

In this way, the prototyped system is able to measure, view, save and recover the principal vital constants of a human-being. Easy-to-use is another of its features which, once the patient knows how the system works, it does not require medical assistance necessarily to put on all the electrodes and sensors as well as to start the software designed to take measures. Moreover, it is small and lightweight which allows to easily carry it to anywhere. In addition to this, the rapidity into which the system takes measures and saves them to make them available for medical evaluation is also another feature to highlight. Above all, efforts have been made to design the cheapest possible system so that it is available for the maximum number of people, no matter their country, race or religion. The only requirement for the good functioning of the system is that the monitoring must be done in an area having Internet coverage.

This system has a hardware which generates patient data, a back-end which measures them, a front-end which shows and retrieves them and a database which saves them.

Concerning the prevention approach, it is based on the presence of mosquitoes and their larvae that can be potentially malaria disease transmitters, always keeping in mind the conditions in which such mosquitoes and their larvae proliferate.

Therefore, its main objective is to determine the location of zones accommodating mosquitoes and their larvae. Once these zones situated, it is about: either alerting the closest population of the existent risk in the zone, releasing infertile male mosquitoes, in case of mosquitoes or, applying the corresponding extermination method, in case of larvae. What is achieved through the detection before taking any action, is to assess the real need of taking such action and to be more effective when the need is real, respecting the ecosystem to the maximum. Indeed, the mosquito itself has a particular role in the ecosystem that must not be underrated, if it is desired to preserve it.

In this way, the prototyped system is able to locate and identify stagnant waters in pictures taken in a certain area as well as to locate, detect and classify mosquitoes from pictures taken also in a certain area. The detection or identification is always carried out with the complementary help of ground and air sensors. A web camera fitted in a drone, in a rover or placed statically, is in charge of taking such pictures. Thus, in case of not having a static camera, after the studied area scanning, what it is finally achieved is a map where the places where mosquitoes and puddles have been detected, are pointed with a pin. Indeed, from this map, the mosquito density in the studied area can be known.

Nevertheless, once puddles have been located, taking advantage of the fact that the land orography does not generally change abruptly, the puddle model can be extracted from the study of a specific puddle of the studied area. This makes the area scanning not necessary anymore and, at the same time, makes this approach very suitable to deal with large distances.

The fact of using this prototyped system has also other benefits. The most important benefit is that by combining Artificial Intelligence for detecting puddles and mosquitoes with environmental data given by ground and air sensors, the probability of successfully detecting increases a lot. Another benefit is that there is no need of a person to locate stagnant waters of a determined area as a drone or a rover can do it instead. Moreover, a drone or a rover allows to sweep either areas difficult to access or vast areas, as well as to do it in a fast and controlled way. Above all and as the previous system, efforts have been made to design the cheapest possible system so that it is available for the maximum number of people.

Nonetheless, the scope of this project is much wider, actually it can be implemented at any place in the world with any disease transmitted by a flying insect such as dengue fever, yellow fever, Zika fever, chikungunya virus, West Nile disease, Japanese encephalitis, sleeping sickness, leishmaniasis, filariasis and lyme disease among others.

In fact, these diseases generally also cause fever, headaches and body aches like malaria does, proving that the diagnostic approach is also perfectly applicable for these cases. Moreover, the first approach can be used to monitor any patient from his or her home, avoiding the need of staying at the hospital for being monitored.

Regarding the second approach, it can be used to fight against mosquitoes, flies or any flying insect near populated zones without the need of fighting especifically against malaria. This makes this approach suitable for any place of the world where there is a problem related to flying insects and it is desired to control and quantify them.

Objectives

After the above introduction and to clearly understand what it is pursued with the elaboration of this project, below the objectives of such project are presented in a summarized way.

- Design of a multiparametric system able to measure, view, save and recover the principal vital constants of a human-being.
- Transmission of all the data generated by the multiparametric system to any place of the world.
- Automated detection of a puddle from a picture through the use of Artificial Intelligence.
- Automated identification of different mosquitoes species from a picture through the use of Artificial Intelligence.
- Design of a sensors system able to gather environmental data from the ground and from the air.
- Integration of the sensors system and the automated recognition system of mosquitoes and puddles in a single architecture.

CHAPTER 1. DEFINITION OF THE MULTIPARAMETRIC HARDWARE

Being conscious of what is the final pursued objective, the following three chapters are going to be focus on the development of a prototyped system able to diagnose such a deadly disease as malaria or dengue in a human being.

The main goal of this chapter is to define the multiparametric hardware, to expose the reasons for having chosen this specific hardware and to describe its physical and technical features as well as its communication protocol.

1.1. Main objective

The multiparametric hardware was designed to display data corresponding to some physiological constants such as the electrical activity of the heart (ECG), the respiration variation (RESP), the temperature variation (TEMP), the peripheral capillary oxygen saturation (SPO2), the pulse rate (PR) and the non-invasive blood pressure (NIBP). These data can be measured by means of an Integrated Multi-Parameter Module (MMP) manufactured by ZUG Medical Systems SAS [14].

1.2. Multiparametric hardware choice

Before choosing the MMP manufactured by *ZUG Medical Systems SAS*, different options were compared to assess their trade-offs and thus justify the final decision.

In this way, some options have been chosen according to two requisites which are: to be an OEM system and to have as main function the measurement of multiple physiologic parameters. Hence, 11 options distributed among 9 companies have been found. To compare them, the parameters considered are: the price and if they support measurements of ECG, SpO2, RESP, TEMP, NIBP, PR (see Table 1.1).

Since the most comprehensive module is being seek, able to measure the most important human physiological constants, all the options not supporting the 6 possible measurements are directly discarded. Moreover, although the PM4100 is a comprehensive module, it has been also discarded as it is not protected against defibrillation and electrosurgery.

Therefore, two options remain which are the MP01000 and Multi-Parameters Module. Now, the factor determining the final decision is the price of the system. As already said, one of the aims of this project is develop a multi-parametric system whose scope is the widest possible, thus, as it seems reasonable, the hardware must be the cheapest allowing more people to take advantage of this technology. Thus, considering these options, the one offering the lowest price is the Multi-Parameters Module manufactured by *ZugMed*.

In addition, there are other features provided by this Multi-Parameters Module that also makes it a very interesting option. For instance, the *ZugMed* hardware due to its compact and light format can be used at any place, adapting some of its functions to the patient's maturity degree. This makes it really suitable in developing countries, for patients staying at

the hospital so to be only monitored or for simply controlling some groups at risk. Moreover, according to the given specifications, it fulfills the medical standards (see Table 1.2).

	ECG	SpO2	RESP	TEMP	NIBP	PR	Price
Berry [1] PM4100 Module	X	X	X	X	X	X	320$ [2]
Bitalino [3] (r)evolution Board Kit BT	X						195.5€[3]
Corscience ChipOX [4] COR12 [5]	 X	X				X	213.70 € 179.50 €
Masimo [6] MX-5 OEM Board		X	X			X	200$ [7]
medlab [8] MP01000	X	X	X	X	X	X	615.30 €
Medtronic [9] Multifunctional Respiratory PCBA OEM Platform		X	X			X	195.5$ [10]
Par Medizintechnik NIBP Development Kit [11] PAR PORT [12]	 X	X	 X	 X	X	X	257.80 € 223.40 €
Vectracor [13] OEM ECG Module	X						193.60 €
ZugMed [14] Multi-Parameters Module	X	X	X	X	X	X	408$ (HW) 250$ (SW)

Table 1.1: Hardware studied options

Std. num.	Name of the standard	Version
YY 1079	”ECG monitor” YY 1079-2008	2008
YY 0782	MEE — part 2-51: particular requirements for the basic safety and essential performance of electrocardiographs	2010
IEC60601-2-2 5	MEE — part 2-25: particular requirements for the basic safety and essential performance of electrocardiographs	2011
IEC60601-2-3 0	MEE — part 2: particular requirements for safety of automatic cycling indirect blood pressure monitoring equipment	1996 A1:1999
EN1060-1	Non-invasive sphygmomanometers — part 1: General requirements	1995
EN1060-3	Non-invasive sphygmomanometers — part 3: Supplementary requirements for electro-mechanical blood pressure measuring systems	1997
ISO80601-2-6 1:2011	MEE — part 2-25: particular requirements for the basic safety and essential performance of pulse oximeter	2011
YY0784-2010	MEE — particular requirements for the basic safety and essential performance of pulse oximeter equipment for medical use	2010

Table 1.2: Standards followed by MMP

1.3. Physical characteristics

The physical characteristics of the MMP will be presented in this Section. First, Section 1.3.1. describes the electrical and environmental specifications. Then, the main components of the hardware are listed in Section 1.3.2.

1.3.1. Electrical and environmental specifications

Globally, the electrical and environmental behavior of this hardware can be defined by Table 1.3 and 1.4.

Supplied voltage	+12V DC, offset voltage between $\pm$10% of voltage full range
Power	$\leq$3 W

Table 1.3: Electrical characteristics

	Work environment	**Storage environment**
Temperature	10 ℃ - 40 ℃	-20 ℃ - 70 ℃
Humidity	15% - 90% non-condensation	15% - 90% non-condensation
Altitude	from -170 m to 1700 m	from -170 m to 1700 m

Table 1.4: Environment characteristics

1.3.2. Components

The MMP can be decomposed in two main components: the communication module and the measurement module (Fig. 1.1). The former is the one in the left top corner (inside the red and dashed square) while the latter is the remaining part.

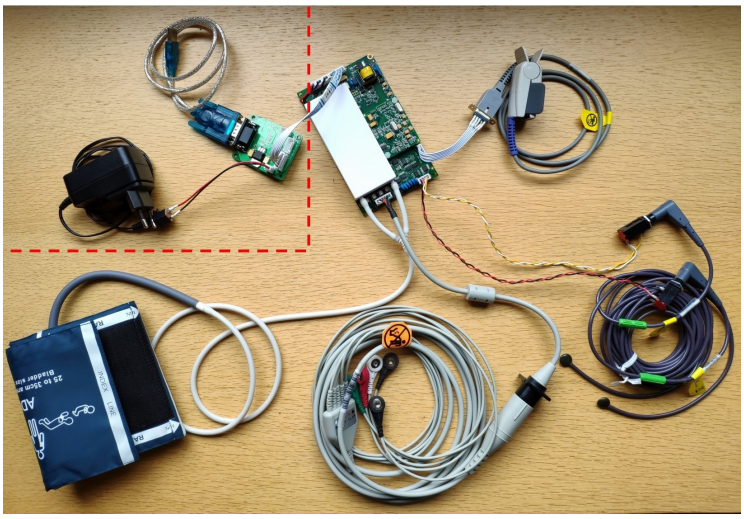


Figure 1.1: Parts of the MMP

1.3.2.1. *Communication module*

Regarding the communication module, it includes a small electronic board, also called serial port adapter plate which acts as a liaison between the host and the measurement module. This serial port adapter allows the MMP to be connected to a PC and, at the same time, to be powered. Thus, this module has an interface of power and communication (Fig. 1.1) which consists of six pins (Fig. 1.2) and behaves such Table 1.5 indicates.

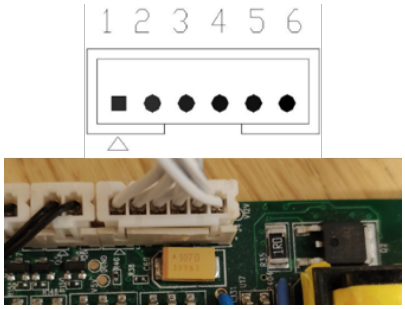


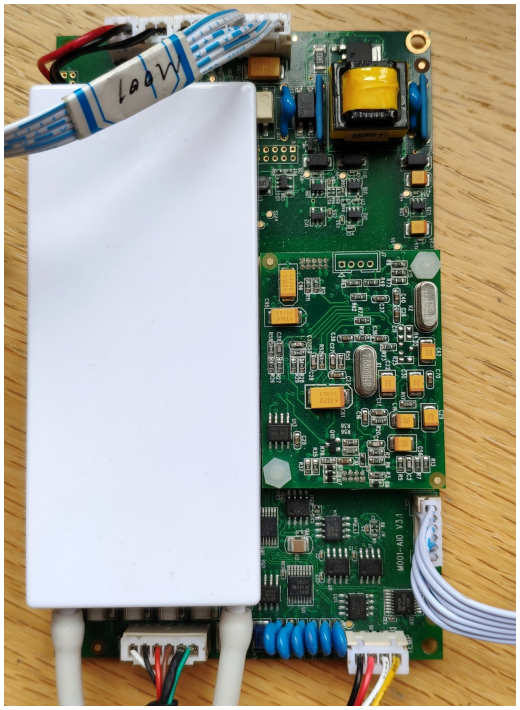
Figure 1.2: Schematic vs physical view of interface socket J4

Pin	Signal	Signal description
1	TXD	Sending data of UART
2	RXD	Receiving data of UART
3	DGND	Ground
4	+12 V	Inputting 12 V power
5	DGND	Ground
6	+12 V	Inputting 12 V power

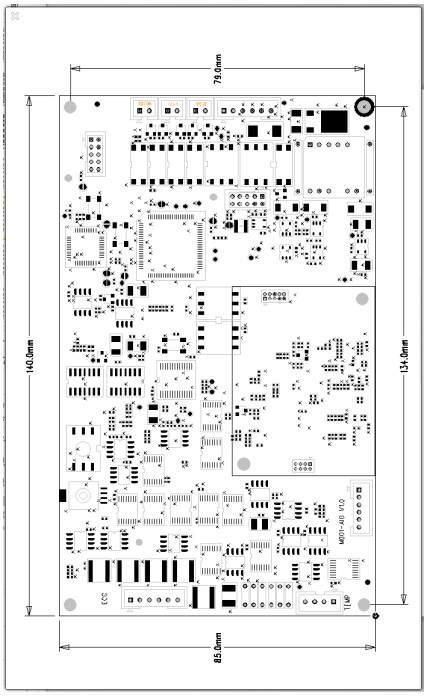
Table 1.5: Definition of interface socket J4

1.3.2.2. *Measurements module*

The measurement module is composed of one electronic board (whose external view corresponds to Figure 1.3(a) and internal view to Figure 1.3(b)), five leads ECG (Fig. 1.4(a)), two temperature sensors (Fig. 1.4(b)), one finger pulse oximeter (Fig. 1.4(c)) and one sphygmomanometer (or blood pressure gauge, Fig. 1.4(d)).

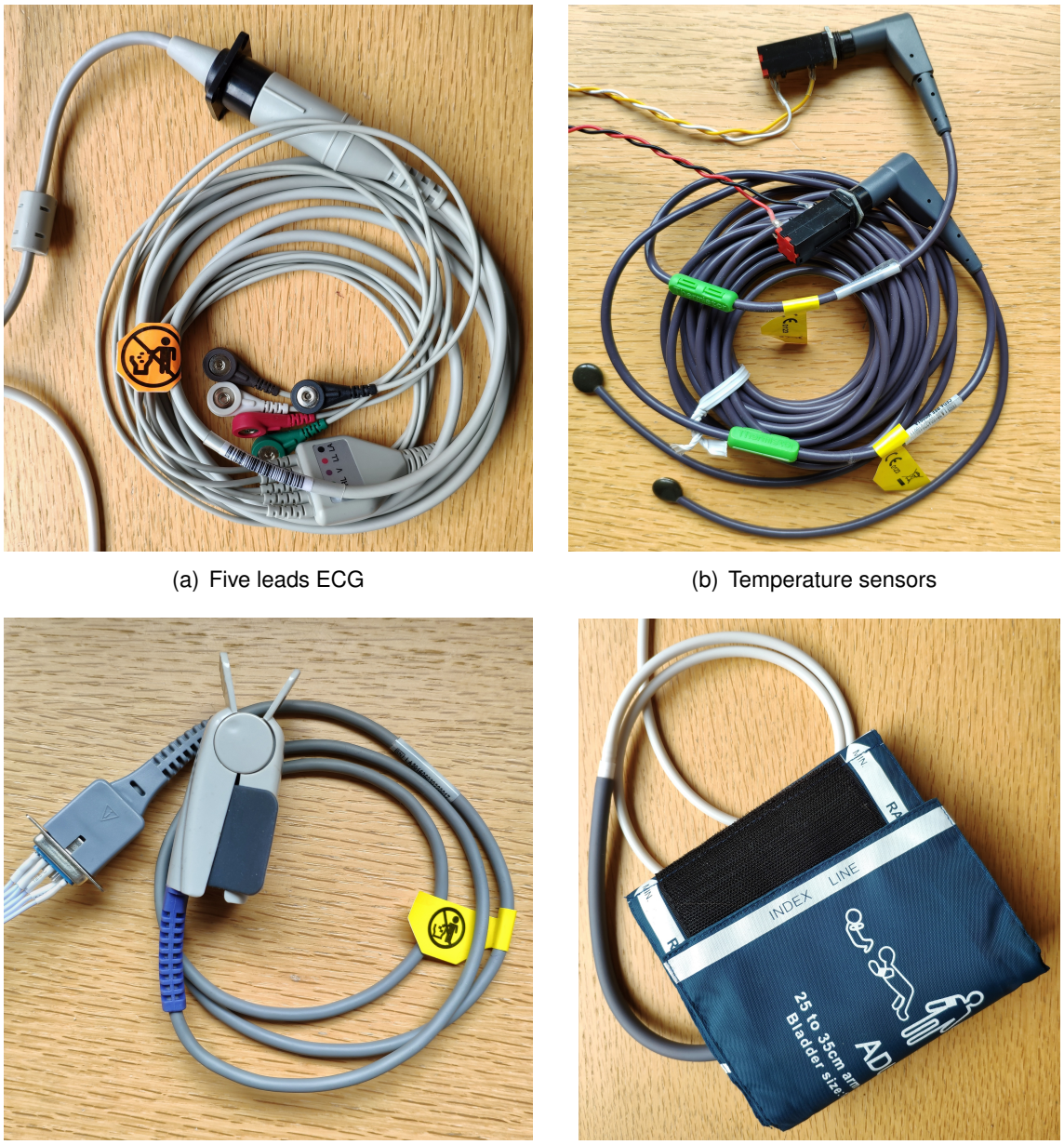


(a) External view



(b) Internal view

Figure 1.3: MMP views



(a) Five leads ECG

(b) Temperature sensors

(c) Finger pulse oximeter

(d) Sphygmomanometer

Figure 1.4: Peripheral elements of the electronic board

Note that there are five ECG electrodes: one for the left arm, one for the right arm, one for the left leg, one for the right leg and one for the chest.

Similar to the communication module, the ECG leads, the temperature sensors as well as the SpO2 sensor have an interface which allows them to be connected to the electronic board of the measurement module and hence, to receive and transmit packets of data. The behavior is summarized in Table 1.6, 1.7 and 1.8 respectively, that is, firstly for the ECG leads, then for the temperature sensors and finally for the SpO2 sensor.

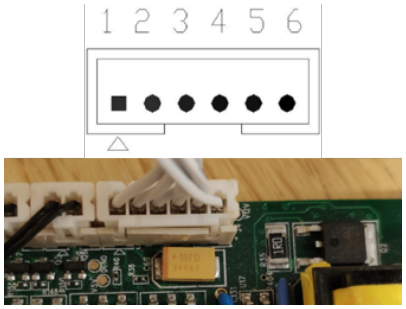


Figure 1.5: Schematic vs physical view of interface socket J10

Pin	Signal	Signal description
1	RA	Right arm
2	LA	Left arm
3	LL	Left leg
4	V1	Chest lead, in n°4 ribs stick to sternum
5	ECG SHIELD	Signal shielding wire
6	RL	Right leg

Table 1.6: Definition of interface socket J10

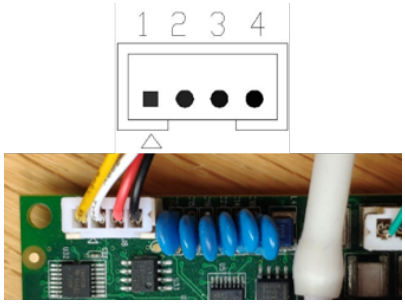


Figure 1.6: Schematic vs physical view of interface socket J8

Pin	Signal	Signal description
1	TEMP1	Temp sensor 1, + input
2	TGND	Temp sensor 1, - input
3	TEMP2	Temp sensor 2, + input
4	TGND	Temp sensor 2, - input

Table 1.7: Definition of interface socket J8

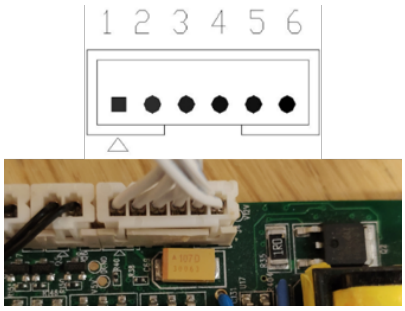


Figure 1.7: Schematic vs physical view of interface socket J5

Pin	Signal	Signal description
1	1_WIRE	Sensor detection signal wire
2	GND	Ground
3	IR	IR LED positive electrode
4	RED	RED LED positive electrode
5	SPO2-	Light signal-
6	SPO2+	Light signal+

Table 1.8: Definition of interface socket J5

1.4. Technical characteristics

Technical characteristics can be explained by the five types of measurements the MMP can carry out simultaneously, that is ECG, respiratory, temperature, SpO2 and NIBP measurements (PR measurements are included in the last two types). Hence, each of these measurement types has its particular method of measurement, its precise function and its specifications, which are going to be described as follows.

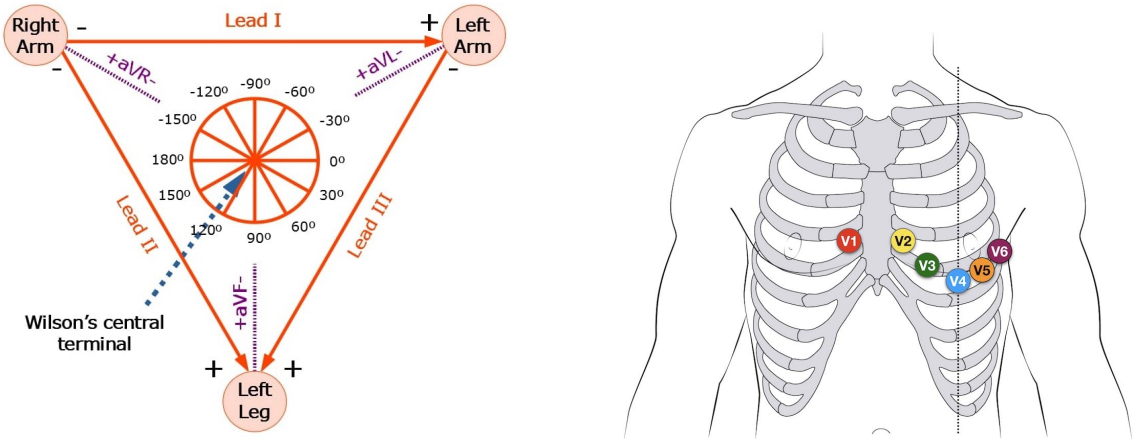
1.4.1. ECG measurement

1.4.1.1. Method of measurement

According to the MMP manual [16], the method of ECG measurement is such that:

> "The heart will generate electrical stimulation, biological electricity and transmit these signals to body surface before the shrinks mechanically. Different parts of the body will generate different electric potential change and form potential difference on body surface. Through recording potential difference will form dynamic curve, the curve is ECG signal."

Concerning the relationship between the leads and the ECG physical electrodes, Table 1.9 shows how the twelve leads of a standard ECG can be obtained from these electrodes. Each of these leads is in charge of picking up electrical activity from a different position on the heart muscle. In fact, the first six leads explained in this Table, named limb leads, can be graphically understood through Figure 1.8(a). Although there are only five electrodes, by moving the chest electrode around the heart, that is from the points V1 to V6 (see Fig. 1.8(b)), the precordial leads can be measured. Note that the right leg electrode is the "grounding electrode"; it is not involved in any leads, but the ECG will not record unless the electrode is attached.



(a) Limb leads

(b) Precordial leads (Source: [15])

Figure 1.8: The 12 ECG leads graphically explained

Lead	Type	Electrode position on the body
Lead I	Bipolar	Right arm electrode negative pole and left arm electrode positive pole.
Lead II	Bipolar	Right arm electrode negative pole and left leg electrode positive pole.
Lead III	Bipolar	Left arm electrode negative pole and left leg electrode positive pole.
Lead aVR	Bipolar	Wilson's central terminal 'negative pole' and right arm positive pole.
Lead aVL	Bipolar	Wilson's central terminal 'negative pole' and left arm positive pole.
Lead aVF	Bipolar	Wilson's central terminal 'negative pole' and left leg positive pole.
Lead V1	Unipolar	It faces the surface of the right ventricle and the septum.
Lead V2	Unipolar	It faces the surface of the right ventricle and the septum.
Lead V3	Unipolar	It faces the anterior wall of the left ventricle.
Lead V4	Unipolar	It faces the anterior wall of the left ventricle.
Lead V5	Unipolar	It faces the lateral wall of the left ventricle.
Lead V6	Unipolar	It faces the lateral wall of the left ventricle.

Table 1.9: Definition of the 12 ECG leads

1.4.1.2. Function

The operation mode of the ECG measurement is summarized in Table 1.10. For further details the reader is referred to [16].

Output	HR, ecg wave, respiratory data, respiratory wave, ST offset value of III, V1 channel and arrhythmia
Indirectly included	Temperature
Patient type	Adult, children, neonate
Measurement mode	Diagnose (0.05 Hz - 130 Hz filter range) Monitor (0.05 Hz - 40 Hz filter range) HARDEST (5 Hz - 20 Hz filter range) Operation (1 Hz - 25 Hz filter range)
Calibration	Record of the amplitude of wave by inputting a standard voltage 1 mV.
Heart rate calculation / arrhythmia analysis channel	I, II, V1 channel can be used as analysis channel or any other channel chosen by yourself
Trapped wave mode	50 Hz, 60 Hz, 50/60 Hz or none (turned off)
Sensor used	Five leads ECG

Table 1.10: ECG measurement function

1.4.1.3. Specifications

Concerning the ECG outputs, they can be graphical or numerical. Up to six graphics can be displayed in parallel (following the specification of Table 1.11) either with the activity of lead I, lead II, lead III, lead aVR, lead aVL, lead aVF, lead V1, lead V2, lead V3, lead V4, lead V5 or lead V6. Indeed, as there are only six graphics, only six of the twelve leads can be viewed in parallel. Regarding the numerical data corresponding to the heart rate, they follow the specification of Table 1.12.

Range	0.15 - 5.5 mV
Accuracy	2.36 μV/LSB
Resolution	1%

Table 1.11: ECG wave specification

Range	15 - 300 bmp
Accuracy	$\pm$1 bmp
Resolution	1 bmp

Table 1.12: HR (heart rate) specification

1.4.2. Respiratory signal measurement

1.4.2.1. Method of measurement

According to the MMP manual [16],

> "The method of respiratory signal measurement is based on the principle of impedance method that is according to the change of thoracic impedance when people breathe and the thorax fluctuating."

1.4.2.2. Function

The respiratory measurement depends completely on the five leads ECG and thus on the ECG measurement.

1.4.2.3. Specifications

Concerning the respiratory outputs, they can be graphical or numerical. Therefore, on the one hand, a graphic displays the respiratory activity according to the specification of Table 1.13. On the other hand, the respiratory rate is also numerically displayed according to the data specification of Table 1.14.

Range	0.15 - 5.5 mV
Accuracy	2.36 μV/LSB
Resolution	1%

Table 1.13: Respiratory wave specification

Range	15 - 120 rpm
Base resistance value	500 - 2000 Ω
Variable resistance value	0.2 - 3.0 Ω

Table 1.14: Respiratory specification

1.4.3. Temperature measurement

1.4.3.1. Method of measurement

The temperature measurement method is based on thermistor.

1.4.3.2. Function

The temperature measurement depends directly on the temperature sensors and the only output corresponds to the current measured temperature. It is worth noting that there will always be two values as there are two independent temperature sensors.

1.4.3.3. Specifications

Regarding the numerical temperature output, it follows the specification of Table 1.15.

Range	0 - 50 ℃
Resolution	0.1 ℃

Table 1.15: Temperature specification

1.4.4. SpO2 measurement

1.4.4.1. Method of measurement

According to the manual of the MMP hardware,

> ”The method of SpO2 measurement is based on arterial blood absorption of impulse type infrared and red spectrum, the photoelectric sensor receives red spectrum and the infrared which are transmitted through finger, then transmits the signal to photocurrent signal amplifier, the signal is processed by the

method as below: voltage amplification, filtering, digitizing, feature recognition and algorithm, so as to obtain the parameters of SpO2 and PR."

1.4.4.2. Function

The operation mode of the SpO2 measurement is summarized in Table 1.16. For further details the reader is referred to [16].

Output	SpO2, PR, PI, transmission of pulse wave signal
Patient type adaptation	Adult, children, neonate
Module status	Real time transmission of hardware status, software status and sensor status in the form of an alarm
Sensor used	Finger pulse oximeter

Table 1.16: SpO2 function

1.4.4.3. Specifications

Similarly to the aforementioned sensors, the outputs of the SpO2 sensors can be graphical or numerical. On the one hand, the SpO2 value, PR and PI are numerically displayed following the specification of Table 1.17, 1.18 and 1.19. On the other hand a graphic is displayed showing the SpO2 activity. However, the MMP developer manual does not provide a concise description of the data displayed in the graphic. For this reason, a validation process of the data will be performed further on.

Range	0 - 100%
Accuracy	70 - 100%, $\pm$2%; $<$ 70%, undefined
Resolution	1%

Table 1.17: SpO2 specification

Range	25 - 250 bpm
Accuracy	$\pm$2%
Resolution	1 bmp

Table 1.18: PR specification

Range	0 - 20%
Accuracy	undefined
Resolution	0.001%

Table 1.19: PI specification

1.4.5. NIBP measurement

1.4.5.1. Method of measurement

According to the manual of the MMP hardware,

> "The method of NIBP measurement is oscillometric, the pressure sensor get pulse wave signal during stepwise deflation, and then the measurement system will obtain results of SBP, MAP, DBP and PR by signal processing and algorithm."

1.4.5.2. Function

The operation mode of the NIBP measurement is summarized in Table 1.20. For further details the reader is referred to [16].

Output	SBP, MAP, DBP and PR
Patient type	Adult, children, neonate
Measurement mode	Manual mode (startup command starts a measurement) Automatic mode (Timing automatic starts the measurement with a cycle time to be selected: 1 min, 2 min, 3 min, 4 min, 5 min, 10 min, 15 min, 30 min, 60 min, 90 min, 2 h, 3 h, 4 h, 8 h) Continuous mode (continuous measurement during five minutes with an interval of 5s between each two measurements.)
Calibration	Through the continuous real time pressure provided by the module
Leakage detection	Funtion to detect if the gas path has a gas leak
Double protection	Over-pressure and over-time protections of hardware and software.

Table 1.20: NIBP function

1.4.5.3. Specifications

Regarding the NIBP outputs, they are numerical and corresponds to SBP, MAP, DBP and PR, which fulfill the specifications of Table 1.21, 1.22, 1.23 and 1.24, respectively. Moreover, although it is not numerically shown, the cuff pressure also belongs to the NIBP system as well as the double protection which follow the specifications of Table 1.25 and 1.26.

Range	40 - 270 mmHg (adult), 40 - 200 mmHg (children), 40 - 130 mmHg (neonate)
Accuracy	Mean deviation $< \pm 5$ mmHg, standard deviation < 8 mmHg
Resolution	1 mmHg

Table 1.21: SBP specification

Range	20 - 230 mmHg (adult), 20 - 175 mmHg (children), 20 - 100 mmHg (neonate)
Accuracy	Mean deviation $< \pm 5$ mmHg, standard deviation < 8 mmHg
Resolution	1 mmHg

Table 1.22: MAP specification

Range	10 - 210 mmHg (adult), 10 - 162 mmHg (children), 10 - 90 mmHg (neonate)
Accuracy	Mean deviation $< \pm$ 5 mmHg, standard deviation < 8 mmHg
Resolution	1 mmHg

Table 1.23: DBP specification

Range	30 - 240 bmp (adult), 30 - 240 bpm (children), 40 - 240 bpm (neonate)
Accuracy	$\pm$2 bpm,or $\pm$2% of the reading (the bigger one)
Resolution	1 bmp

Table 1.24: PR specification

Range	0 - 300 mmHg
Accuracy	$\pm$2%,or $\pm$1% of the reading (the bigger one)
Resolution	1 mmHg

Table 1.25: Cuff pressure specification

Over voltage protection	$< 297 \pm 3$ mmHg (adult), $< 250 \pm 3$ mmHg (children), $< 150 \pm 3$ mmHg (neonate)
Max time of pressure measurement	$<$ 120 s (adult), $<$ 120 s (children), $<$ 90 s (neonate)

Table 1.26: Double protection specification

1.5. Communication protocol

The MMP communicates with a host or PC using a specific protocol. This communication consists of transmitting and receiving commands carrying out operations, returning data and notifying the system status.

1.5.1. Serial port configuration

The data is transferred through a serial port which must fulfill the settings listed in Table 1.27.

Logic level	TTL
Data format	Start bit + 8 data bits + 1 stop bit
Checksum	No check bit
Baud rate	115200 bps

Table 1.27: Serial port settings

1.5.2. Data format

To be able to establish a dialogue between the host and the MMP, there must be an agreement between these two components regarding the transmitted and received data format, defined by the communication protocol (see Table 1.28).

Name	Length (byte)	Description
Start character	1	It is used to look for the beginning of the packet when it is parsed. Its constant is 0xFA.
Packet length	1	It is equal to the number of bytes from the packet start character to the checksum.
Parameter type	1	It corresponds to the packet identification of different parameters, that is, 0x01 stands for ECG (which includes temperature and respiration), 0x02 stands for NIBP, 0x03 stands for SpO2.
Packet type	1	It is used to identify the different packet types. There are four types: Control command packet (DC), Requesting command packet (DR), Command answer packet (DA) and Common data packet (DD).
Packet ID	1	It is used to identify the packet ID.
Serial number	4	It is used to distinguish which command answer packet (DA) responds to which control command packet (DC) or requesting command packet (DR). The serial number of DD is increased one by one, in order to avoid packet loss and analysis when common data packet (DD) is sent.

Name	Length (byte)	Description
Data content	Undefined	Specific format and length depends on the packet ID.
Checksum	1	Considering the time consuming of CRC calculation, the checksum is calculated by the method of single-byte reduction. The calculation includes the packet length, the parameter type, the packet type, the packet ID, the serial number, data contents except the start character.

Table 1.28: Data packet format

CHAPTER 2. DESIGN OF THE MULTIPARAMETRIC SYSTEM

The main purpose of this chapter is to describe and discuss the different parts of the multiparametric system to then zoom out and look at them acting interdependently as an entire system.

2.1. Main aim

The multiparametric system consists of three elements (a back-end, a front-end and a database) and is designed to generate, display, save and retrieve physiological data of a certain patient in a *Windows* environment. The principal idea is to have an intuitive and easy-to-use system working for patients as well as for medical professionals, that is, a double perspective system.

Accordingly, it is strongly recommended to have a look at A.1. before reading this Section.

2.2. *C++* Back-end

Developed with *C++* language, the main function of the back-end is to generate data and to send them to the front-end. The MMP hardware is sold with a program and its private code both developed in *C++* language. Using the hardware, this program is already able to process the sensors measurements and display the physiological constants of a person. However, it only measures physiological data, that is why its code has been modified so to be able to send them to the front-end in order to finally save them in a database. Besides, these modifications also allow the front-end to send some specific commands to the back-end to start taking measurements.

2.2.1. Enhancement of the initial software

The enhancement of the initial software consists of two steps: the creation of a new source code file with its header file, and the modification of the original code provided by the seller in order to match the new features included in the first step.

2.2.1.1. New source code file creation

To accomplish this task, a source code file called ”DataRecording.cpp” with its corresponding header file named ”DataRecording.h” has been added to the original private code. The objective of these two files is to help processing the data that is going to be sent to the front-end and that is going to be received by the file user from the later one. The functions declared in the header file, and developed in the source file, are:

- void setIpAndPort(string ipAndPort): It sets the ip and port of the database.

- void setUsername(string username): It sets the name of the username that is carrying out the measurements.
- void setPassword(string password): It sets the password of the user that is carrying out the measurements.
- void setSchemaName(string schemaName): It sets the used schemaname of the database. The schema name is the name of the whole database containing multiple tables.
- void setPatientId(string patientId): It sets the patientid that is carrying out the measurements.
- void createFolder(string dataId): It creates a folder whose name is the data id of the corresponding set of measurements, into the folder ”MMP Data” located in the root directory of the PC. The folder should contain two files: the trace file and the configuration file. For both files, the content is only informative and allows to identify any problem during the execution. The first file corresponds to the logs of the execution; the second file reproduces the configuration that was used for a particular run.
- void setMinimized(string isMinimized): This function is used to specify whether if the back-end should be run minimized (as a windows service) or not. The input corresponds to a string according to what is sent from the back-end.
- void setClientConnexion(SOCKET client): It sets the established socket connection.
- string getIpAndPort(): It returns a string with the ip and port of the database.
- string getUsername(): It returns a string with the name of the username that is carrying out the measurements.
- string getPassword(): It returns a string with the password of the user that is carrying out the measurements.
- string getSchemaName(): It returns a string with the used schemaname of the database.
- string getPatientId(): It returns a string with the patientid that is carrying out the measurements. The patientid allows to identify the user numerically.
- string getDataId(): It returns a string with the id identifying the generated data.
- string getFolder(): It returns a string with the location of the folder whose name is the data id corresponding to a set of measurements.
- boolean isMinimized(): It indicates if the back-end should be run minimized (as a windows service) or not.
- SOCKET getClientConnexion(): It returns the established socket connection.
- ofstream setRecordTypeLogFile(string type): This function creates a log file whose name is the ”type” input. It has been used basically to create a configuration file in which the applied configuration is written. This file is saved into a folder whose name is its corresponding data id. The function output is the log file itself.

- ofstream setRecordTypeTraceFile(): This function creates a log file called ”Trace.txt” where the program writes all the logs encountered. This file is saved into a folder whose name is its corresponding data id. The function output is the log file itself.
- void gettimeofday(struct timeval * tp, struct timezone * tzp): This function obtains the current time, expressed as seconds and microseconds since the Epoch, and stores it in the timeval structure pointed to by tp. The resolution of the system clock is unspecified. If tzp is not a null pointer, the behavior is unspecified.
- inline string setLongDate(): This function returns a string with the current date time with the following format: DD/MM/YYYY HH:MM:SS,UUU
- void setRecordStringSameLine(const char * type, CString text): This function adds the inputted string by ”text” in the same last line of the file whose name is specified by the ”type” input.
- void setRecordStringNextLine(const char * type, CString text): This function adds the inputted string by ”text” below the last line of the file whose name is specified by the ”type” input.

2.2.1.2. Original code modification

Besides creating these new code files, some code has been added to the original files.

By execution order, the first file modified is the one whose name is ”MainDlg.cpp” which is the main file from which the program is run. Therefore, the first applied modification and one of the first things the program does, is to wait for the database IP, database port, database schema name, user name, user password, patient id and data id. Until these data sent through a TCP/IP socket do not arrive, the program does not continue.

Once the expected data is received, the program sets the configuration from which the measurements are going to be performed and here the second modification is introduced. Hence, in ”Multi_ParaMonitorDlg.cpp”, instead of always setting the same configuration by default, the program establishes a connection with the *MYSQL* database and extracts the configuration already saved into it according to the data id that was sent previously. Notice that this configuration is adjusted and saved by the user from the front-end.

Immediately after, the program can start generating data and here the third and the widest modification is applied. Affecting to the source codes generating numerical data, called ”NibpInfo.cpp”, ”SpoO2Info.cpp”, ”StaticEcgInfo.cpp”, ”StaticRespInfo.cpp”, ”StaticTemp-Info.cpp” using ”DataRecoding.h”, each time data are generated, they are automatically sent to the front-end through the TCP/IP socket with its associated date time.

Regarding the source codes generating signal data, that is the ones named ”SpO2- Wave.cpp” and ”StaticWave.cpp”, the data cannot be sent right after the measuring the vital variables because a preprocessing is required. This preprocessing consists of building a string with all the data previously gathered in a vector. As the amount of signal data is really high, the way of creating this string has been optimized by using some exsiting functions (see ref.[17]) proved to be three times faster than the well-known ”itoa()” *C* function (see ref.[18]). Thus, this function enhances the transmition speed of the data thorugh sockets.

All these modifications allows the back-end to gather as well as to send the generated data by the MMP hardware.

2.2.2. Execution ways

As it is a back-end, the user interface (UI) is not of interest to the user of the whole system, that is why the front-end includes the possibility of executing the back-end as a *Windows* service and thus, to be invisible for the user. The instructions to set the back-end as a Windows Service are given in the appendix A.2.. Furthermore, the front-end also allows the user to execute the back-end as a simple executable in such a way that the UI is displayed. Notice that the UI preserves the same look with or without the modifications presented in Section (fig 2.1).

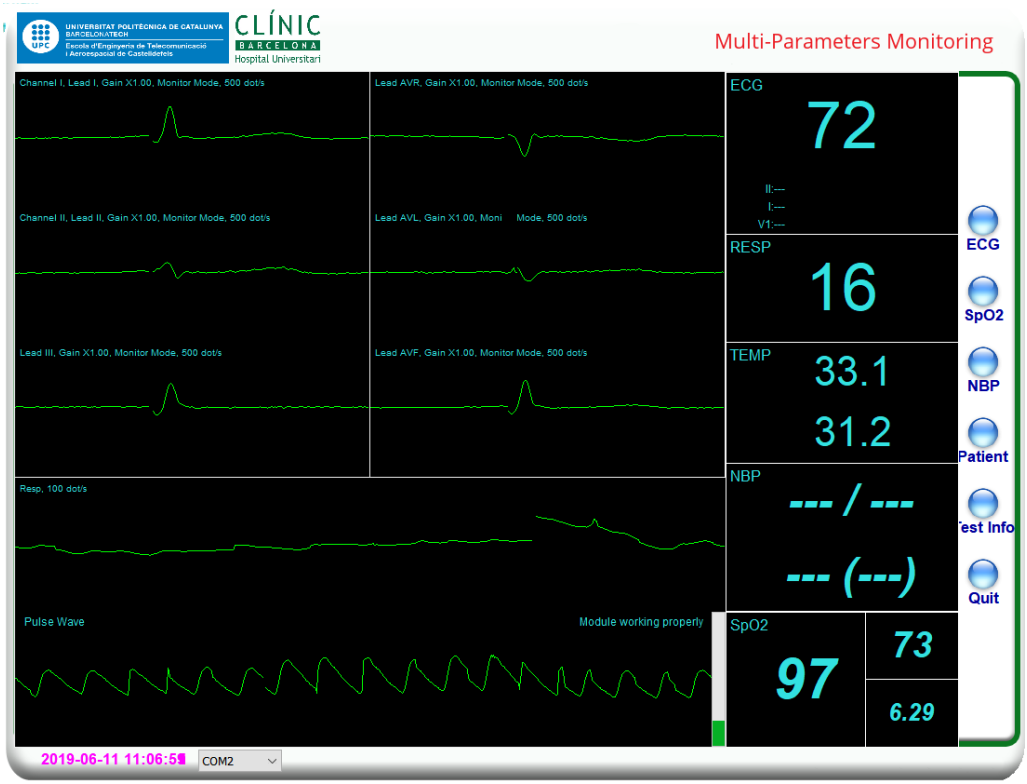


Figure 2.1: View of *MMP* program

2.3. *MySQL* Database

For the system design, a database is required so to save all the data sent from the front-end and to return the requested data by the front-end or the back-end as well. In this case, the one chosen is the *MYSQL* Community Edition database.

The steps to install the *MYSQL* database are thoroughly detailed in Appendix A.3..

2.3.1. Reasons for chosing *MYSQL*

As said in the reference [19], *MySQL* is the most popular Open Source Structured Query Language (SQL) database management system so far, and it is permanently developed, distributed, and supported by Oracle Corporation. In addition, *MySQL* databases are re-

lational, meaning that they store data in separate tables rather than putting all the data in one big storeroom.

As it is a open source management system, it contains a large amount of contributed *MySQL* software such a server, router or workbench. In this case, the server, which is the most important part for a database as it handles all the data, is very fast, reliable, scalable, and easy to use. Furthermore, the *MYSQL Database Server* works in both client/server and embedded systems and the intuitive workbench ease the database management.

All these features encouraged the use of *MYSQL* as a database management system.

2.3.2. Database structure

The created database is called ”mmp_settings” and contains up to 21 different tables, however some of them have the same internal structure. Thus, they can be split up in two groups: the tables related to the configuration of the multiparametric system and the ones related to the data generated by the back-end.

2.3.2.1. MMP system configuration tables

Regarding the tables handling the configuration of the multiparametric system, there are a total of 5 tables with different internal structure whose names are: ”generalsettings”, ”patientdatarelation”, ”ecgsettings”, ”spo2settings”, ”nibpsettings”. Therefore, in Table 2.1, 2.2, 2.3, 2.4 and 2.5, the structure of each is discussed.

Attribute	**Type**	**Description**
patientId	INT(11)	It is the primary key and and the id of the patient.
username	VARCHAR(45)	It corresponds to the virtual name of the user.
password	VARCHAR(45)	It corresponds to the password of the user.
name	VARCHAR(45)	It corresponds to the name of the user.
surnames	VARCHAR(45)	It corresponds to the surnames of the user.
patientType	VARCHAR(45)	There are three types: patient, doctor, nurse
institutionId	VARCHAR(45)	It corresponds to the id of the institution carrying out the measurements.
ecgMeasure	VARCHAR(45)	It can take 2 values: true or false indicating if the ecg values should be taken into account.
spo2Measure	VARCHAR(45)	It can take 2 values: true or false indicating if the ecg values should be taken into account.
nibpMeasure	VARCHAR(45)	It can take 2 values: true or false indicating if the ecg values should be taken into account.

Table 2.1: Internal structure of the ”generalsettings” table

Attribute	Type	Description
dataId	INT(11)	It is the primary key and corresponds to the id of the set of measurements, which is unique for the whole database.
patientId	INT(11)	It corresponds to the id of the patient that is carrying out the set of measurements with the given data id.
recordingDate	VARCHAR(45)	It contains the date and time from which the set of measurements is being saved.

Table 2.2: Internal structure of the ”patientdatarelation” table

Attribute	Type	Description
dataId	INT(11)	It is the primary key and corresponds to the id of the set of measurements, which is unique for the whole database.
ecgLeadMode	INT(11)	There are two ecg modes: 3 leads or 5 leads.
ecgWaveSpeed	INT(11)	The speed of the ecg wave can be set to: 100 dots/s, 250 dots/s, 500 dots/s.
respWaveSpeed	INT(11)	The speed of the respiratory wave can be set to: 50 dots/s, 100 dots/s, 250 dots/s, 500 dots/s.
ecg1mvRef	VARCHAR(45)	It can take 2 values: true or false indicating if the ecg 1 mV should be taken into account.
ecgGain	INT(11)	The gain of the ecg wave can be multiplied by 0.25, 0.5, 1, 2.
ecgFilterMode	VARCHAR(45)	The ecg filter has four modes: Diagnostic,Monitor, HARDEST, Surgery.
ecgTrapMode	INT(11)	The ecg trap mode can be set to: 50 Hz, 60 Hz, 50/60 Hz, none.
ecgHrChannel	VARCHAR(45)	The HR channel can be set to: channel I, channel II, automatic.
respApneaTime	INT(11)	The respiratory apnea time can be set to: 5 s, 10 s, 15 s, 20 s.
respLead	VARCHAR(45)	There are two leads: lead I or lead II.
respSensitivity	INT(11)	It can go from 1 to 5 (the highest the more sensitive and less time is required to capture data).
ecgPace	VARCHAR(45)	It can take 2 values: true or false indicating if the ecg pace should be applied.

Table 2.3: Internal structure of the ”ecgsettings” table

Attribute	Type	Description
dataId	INT(11)	It is the primary key and corresponds to the id of the set of measurements, which is unique for the whole database.
spo2Sensitivity	INT(11)	It can go from 1 to 5 (the highest the more sensitive and less time is required to capture data).

Table 2.4: Internal structure of the ”spo2settings” table

Attribute	Type	Description
dataId	INT(11)	It is the primary key and corresponds to the id of the set of measurements, which is unique for the whole database.
nibpStartMeasure	VARCHAR(45)	It can take 2 values: true or false indicating if the nibp measure has started or not.
nibpStopMeasure	VARCHAR(45)	It can take 2 values: true or false indicating if the nibp measure has been stopped or not.
nibpMeasureMode	VARCHAR(45)	There are three nibp meausre modes: Manual, 5 min continuous measurement, Automode.
nibpMeasure-ModeAutoTime	VARCHAR(45)	This attribute only has a sense when the measurement mode is Automode and can be set to: 1 min, 2 min, 3 min, 4 min, 5 min, 10 min, 15 min, 30 min, 60 min, 90 min, 2 h, 3 h, 4 h, 8h.
nibpPrecharging-PresAdult	INT(11)	The precharging pressure of an adult can be set to: 80, 100, 120, 140, 150, 160, 180, 200, 220, 240, 250, 260, 270 or 280 mmHg.
nibpPrecharging-PresChild	INT(11)	The precharging pressure of a child can be set to: 80, 100, 120, 140, 160, 180 or 200 mmHg.
nibpPrecharging-PresNeonate	INT(11)	The precharging pressure of a neonate can be set to: 80, 100 or 120 mmHg.
nibpStartLeakTest	VARCHAR(45)	It can take 2 values: true or false indicating if the leak test has started or not.
nibpStart-Venipuncture	VARCHAR(45)	It can take 2 values: true or false indicating if the venipuncture has started or not.
nibpVenipuncture-PresAdult	INT(11)	The venipuncture pressure of an adult can be set to: 22, 32, 42, 52, 62, 72, 82, 92, 102, 112 or 122 mmHg.
nibpVenipuncture-PresChild	INT(11)	The venipuncture pressure of a child can be set to: 22, 32, 42, 52, 62, 72 or 82 mmHg.
nibpVenipuncture-PresNeonate	INT(11)	The venipuncture pressure of a neonate can be set to: 22, 32, 42 or 52 mmHg.
nibpSoftware-PresProtection	VARCHAR(45)	It can take 2 values: true or false depending on if the pressure protection is activated or not.
nibpPressure-Calibration	VARCHAR(45)	It can take 2 values: true or false indicating if the calibration should start or not.

Table 2.5: Internal structure of the ”nibpsettings” table

2.3.2.2. MMP system data tables

Concerning the tables containing data generated by the back-end, thus by the MMP hardware, there are 13 tables which can be further divided in two subgroups: the tables containing graphical data and the ones containing numerical data.

In the first subgroup, there are 8 tables corresponding to the 8 different graphics that the back-end can produce, which are named: ”ecg0wave”, ”ecg1wave”, ”ecg2wave”, ”ecg3wave”, ”ecg4wave”, ”ecg5wave”, ”respwave” and ”spo2wave”. Regarding the second subgroup, it consists of 5 tables matching the 5 different numerical outputs the back-end can generate, which are called: ”ecg”, ”resp”, ”temp”, ”spo2”, ”nibp”. Although the data saved in tables of different subgroup differ considerably in size, the internal structure of these tables remains the same. Thus, the structure is composed of 4 attributes whose function is described in Table 2.6. Note that the varchar length of the ”data” attribute value contains an X, meaning that depending on the table, it can vary from 45 to 4500.

Attribute	Type	Description
id	INT(11)	It is the primary key and it has an unique value for each table, that is, the value can be repeated from a table to another but not in a same table
dataId	INT(11)	It corresponds to the id of the set of measurements, which is unique for the whole database
dataPosition	VARCHAR(45)	It allows to know in which position the saved data were generated with respect to the other data of the same set of measurements or dataId
data	VARCHAR(X)	It contains the data itself with its date and time

Table 2.6: Internal structure of tables containing data generated by the MMP hardware

Finally, there are 3 tables whose names are ”additionalecg”, ”additionalimpedance” and ”additionalppg”, which contains the data generated by other types of hardware (different from the MMP hardware). It is worth noting, however, that their structure is the same as the tables containing data generated by the MMP (Table 2.6).

All in all, this is the structure the database should have for the multiparametric system.

2.4. *LabVIEW* Front-end

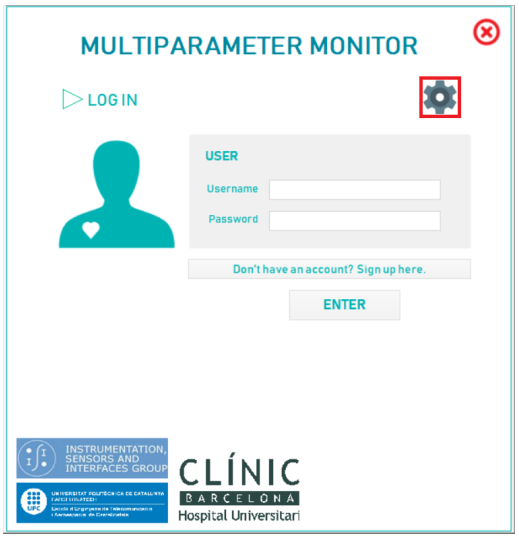
Developed with *LabVIEW* software, the role of the front-end can be summarized in two functions: the first one is to display and save the physiological data from a patient that are being generated by the back-end, and the second one is to retrieve and display already generated data from a patient. Consequently, the first function has been designed according to the patient needs and the second according to the doctor needs. Thus, the front-end has two views designed according to the needs of the user profiles using the application.

In the next Sections, first the configuration of the front-end and the steps to create a new user will be described. Then, once logged with the appropriate credentials of the patient and doctor users, the two possible layouts of the front-end will be described.

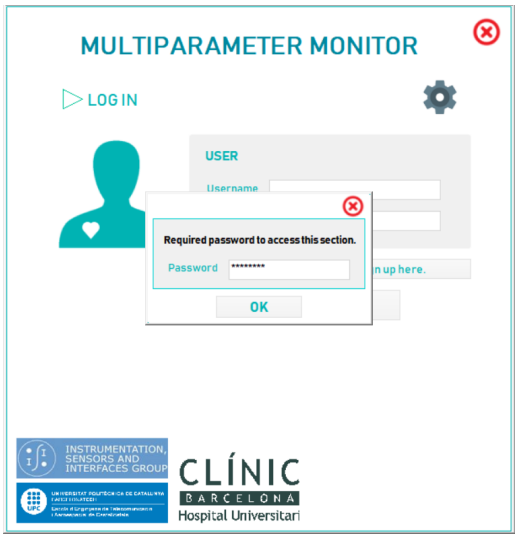
2.4.1. Configuring *MMP_LabView.exe*

After running *MMP_LabView.exe*, the first step is to link it to *MYSQL* database as well as to the back-end *Multi_Para Monitor.exe*. To do so, by clicking on the setting icon (Fig. 2.2(a)) will lead to a settings window (Fig. 2.2(b)). In this window, all the parameters related to the database and the back-end program must be correctly set. After accepting the changes, they will be saved into a ini file called *MMP_LabView_Config.ini* (Fig. 2.2(d)) in such a way that this precise configuration will be remembered the next time the front-end is opened.

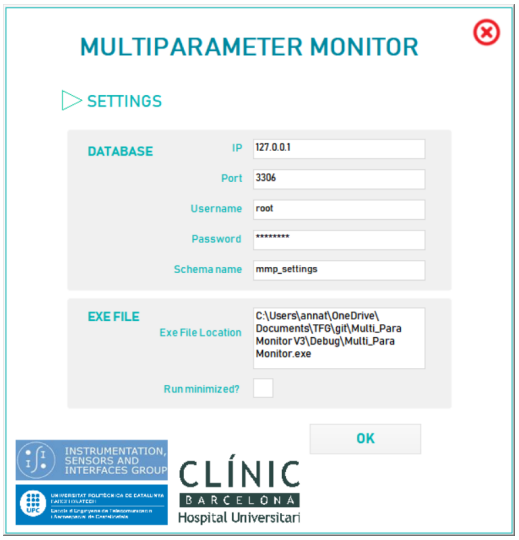
To change the application configuration, a password is required (Fig. 2.2(c)) and by default, it is ”password”; however, it can be changed by editing the value of the *exe_password* field of *MMP_LabView_Config.ini*. This password is unique for the whole front-end, that is every time a password is needed to enter a section, it is always the same.



(a) Settings icon in log in screen



(b) Required password to access settings screen



(c) Settings screen

```
[Section 1]
db_ip = "127.0.0.1"
db_port = "3306"
db_username = "root"
db_password = "password"
db_schema = "mmp_settings"

[Section 2]
exe_file_location = "C:\\git\\Multi_Para_Monitor_x32\\Debug\\Multi_Para Monitor.exe"
exe_run_minimized = TRUE
exe_password = "password"
```

(d) INI Config file

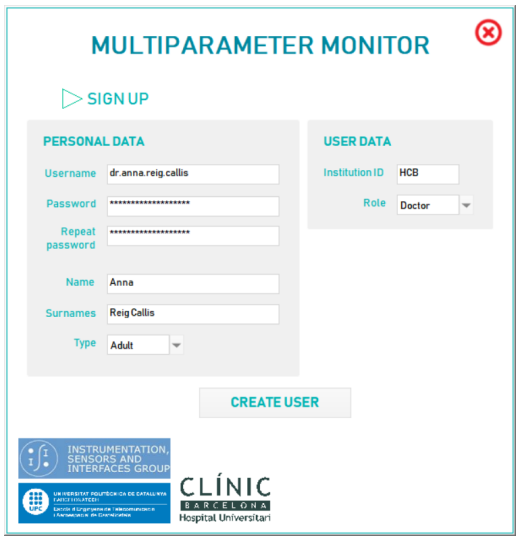
Figure 2.2: Steps to change application settings

2.4.2. Creating a new user

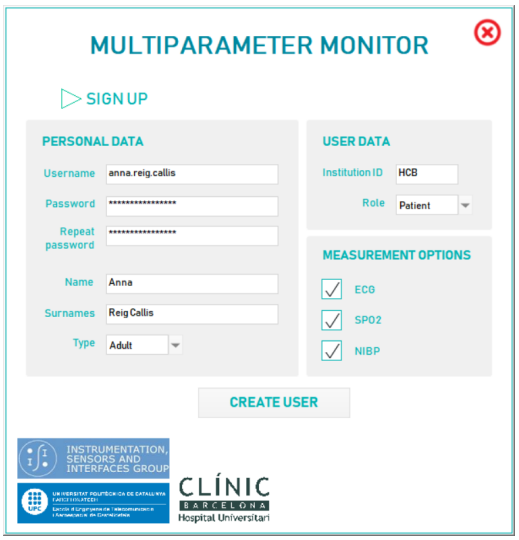
Once arrived here, it is assumed the introduced configuration is the correct one. Thus, a new user can be created.

So, when clicking on *Don't have an account. Sign up here.* in the log in screen (Fig. 2.2(a)), a new window should pop up allowing to create a new user (Fig. 2.3).

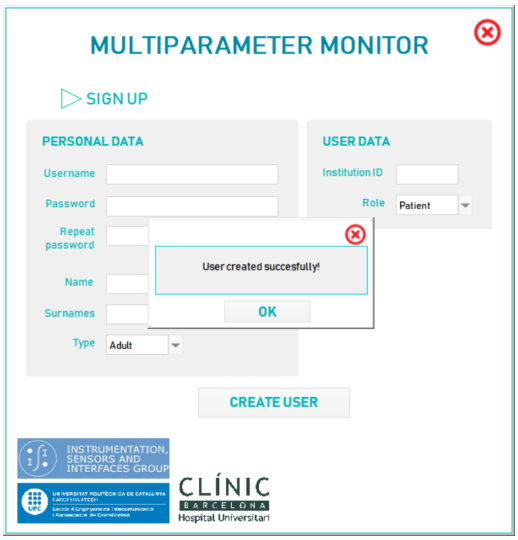
Since this application can work in two different ways (for patients and for medical professionals), two types of profiles can be generated (Fig. 2.3(a) and Fig. 2.3(b)). The difference between a doctor or nurse profile and a patient profile is that the former cannot choose the kind of measurements to execute while the latter can. This can be explained by the simple fact that a doctor or a nurse profile will only replay previous patients measurements which will have the kind of measurements already set.



(a) Doctor profile



(b) Patient profile



(c) Message after a successful new user creation

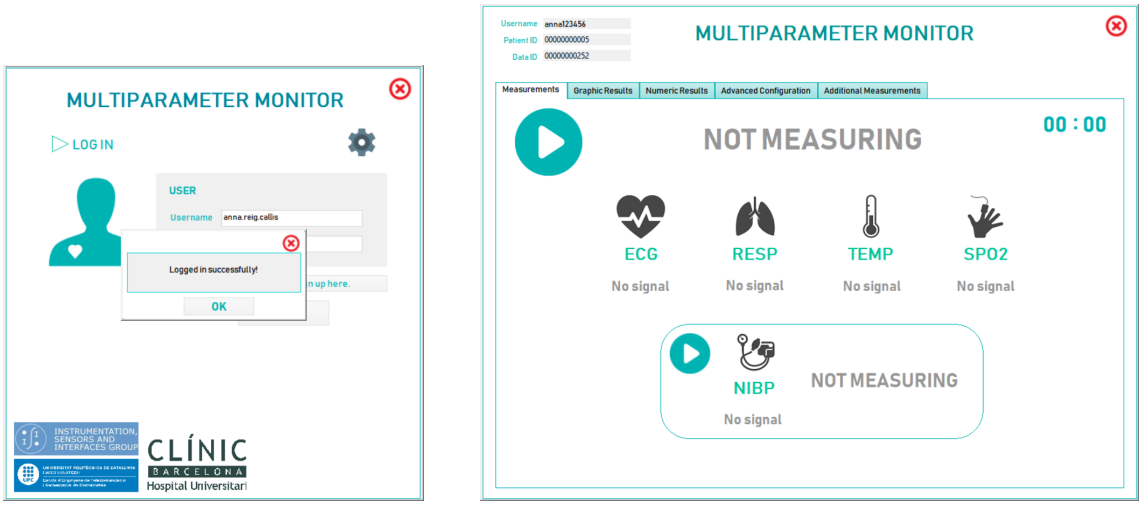
Figure 2.3: Steps to change application settings

All the fields appearing in blank must be filled so to successfully save the new user. Otherwise, the new user won't be created and could not be logged in. Last, the user name must be unique and the password must contain at least 8 characters.

2.4.3. Patient's view

Up to now the two kind of possible users have been created. Thus it is time to log in with them to know what will be the front-end behavior according to the profile type.

Let's start logging in with the patient profile. After introducing the user name and the password correctly (see Fig. 2.4(a)), a new window allowing the user to start measuring his physiologic constants will pop up (see Fig. 2.4(b)). As shown in Figure 2.4(b), this window is quite complex as it is composed of five tabs.

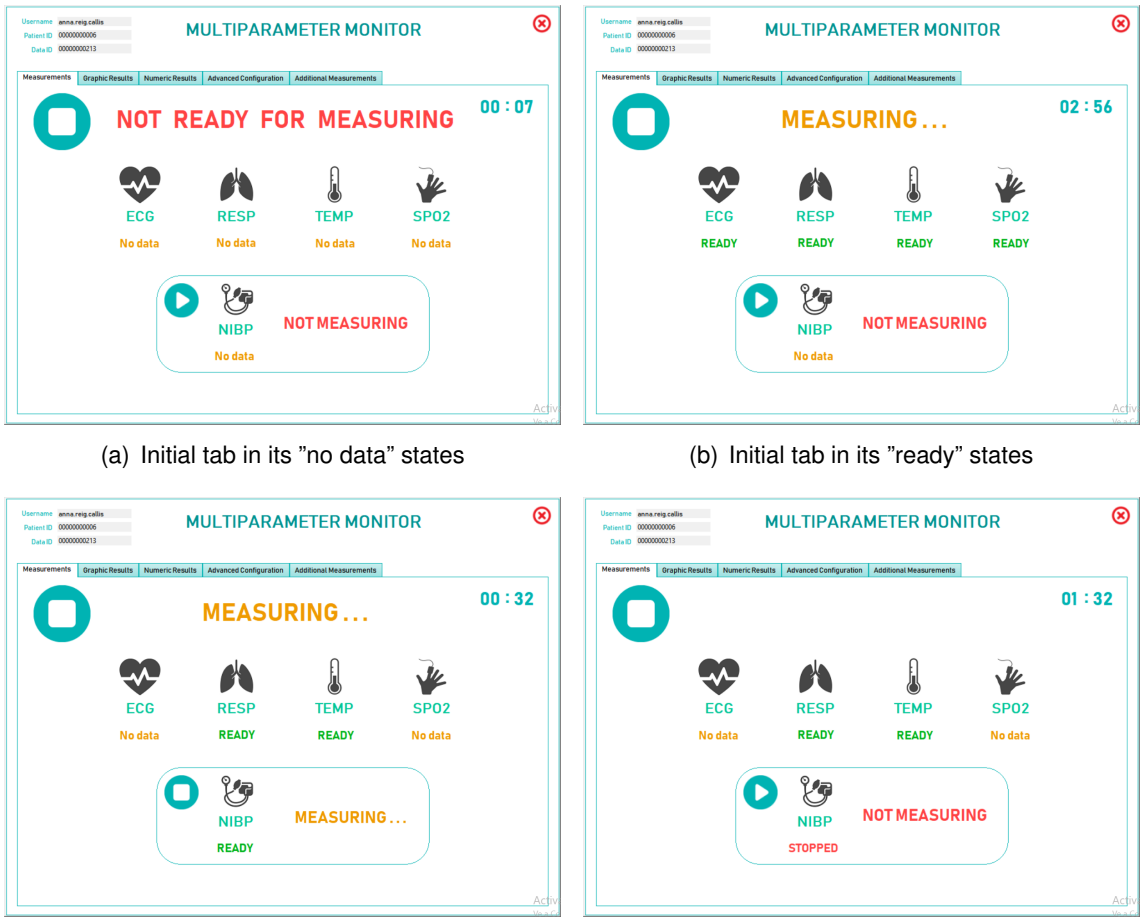


(a) Log in screen

(b) Front-end initial tab in its initial state

Figure 2.4: Logging in with a patient profile

The first tab, called "Measurements", advises the user about the state of the measurements. In fact, each type of measurement has three states: "no signal" when the hardware is not well plugged or it is not answering (see Fig. 2.4(b)), "no data" when the hardware is working well but no data are still obtained (see Fig. 2.5(a)) and "ready" when some data are already saved into the database (see Fig. 2.5(b)). In the case of the pressure, there are two more states: "done" when the pressure has been taken (see Fig. 2.5(c)) and "stopped" when the user has decided to stop the measurement (see Fig. 2.5(d)). Furthermore, for this last case, the "ready" state indicates when the order of taking the pressure has been sent to the back-end program. Moreover, by clicking on the play button, the measurements will start. In the same way, by clicking on the stop button, the measurements will stop.



(a) Initial tab in its ”no data” states

(b) Initial tab in its ”ready” states

(c) ”ready” nibp state

(d) ”stopped” nibp state

Figure 2.5: States of the front-end initial tab

The second tab, named ”Graphic Results”, shows the behavior of the graphics related to ECG, respiration and SpO2 (see Fig. 2.6) which fulfill the specifications defined in the previous chapter. In a point of fact, to avoid getting the user nervous, this tab requires a password to watch the evolution of the signals. Concerning the password, it is the same as the one asked for when accessing the settings window. So, by default, it corresponds to ”password” and can be changed as explained before by editing *MMP_LabView_Config.ini*.

The third tab, called ”Numerical Results”, also acts in an informative way for the user and at the same time displays numerical data such as: HR in beats per minute, RESP in respiration per minute, TEMP in Celsius degrees, SpO2 in percentage, PR in beats per minute, PI in percentage, SBP in mmHg, MAP in mmHg, DBP in mmHg (see Fig. 2.7). Hence, by observing how the data of this tab change, the user will be certain that the system is working as expected. Additionally, as the graphics, these numerical data also fulfills the specifications presented in Section 1.4..

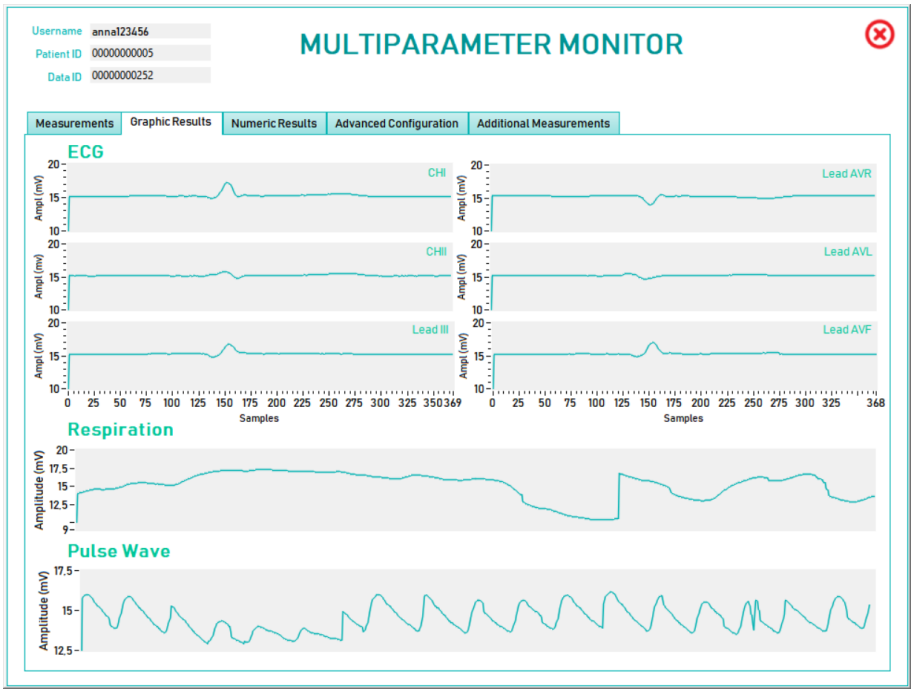


Figure 2.6: ”Graphic Results” tab view

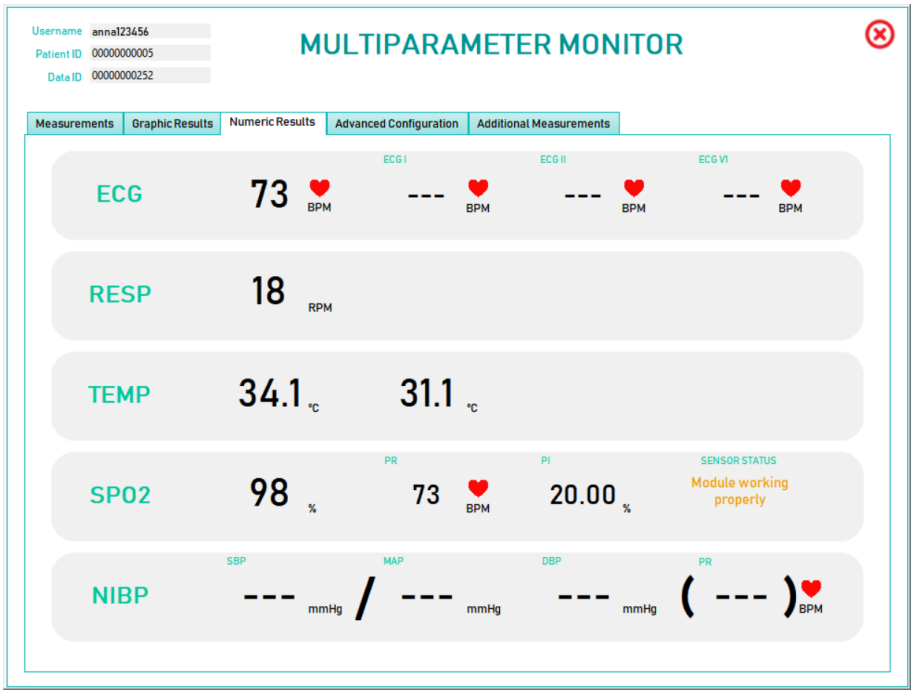


Figure 2.7: ”Numerical Results” tab view

The fourth tab, named ”Advanced Configuration”, also needs a password to enter so to avoid that the patient changes the configuration set by the medical professionals. This tab, as its name already suggests, allows the doctor or the nurse to adjust the system according to the type of patient (see Fig. 2.8). It should be noted that this configuration is linked to the username and appropriately saved into the database.

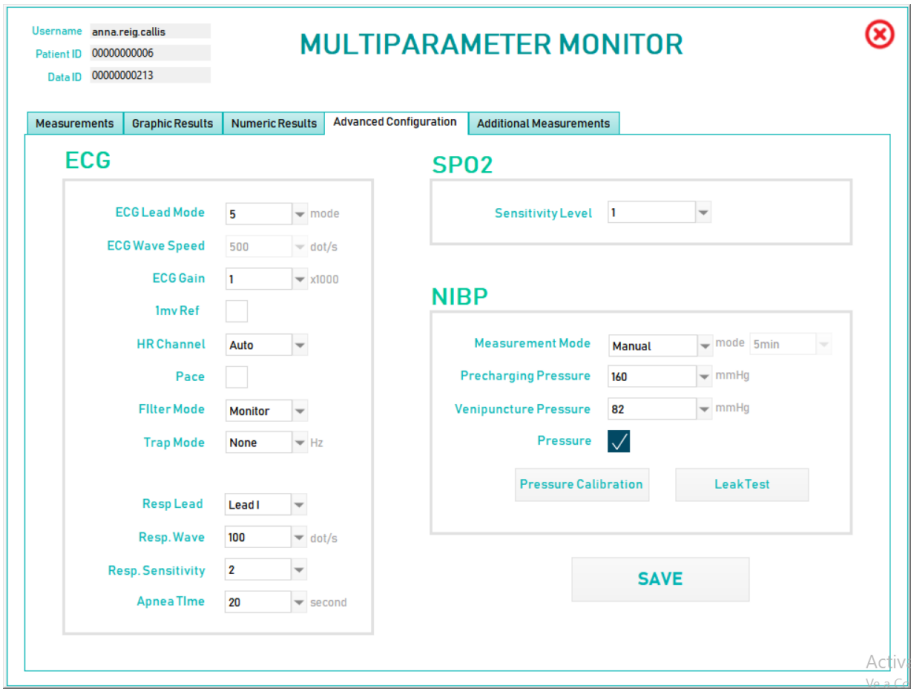


Figure 2.8: ”Advanced Configuration” tab view

The last tab, called ”Additional Measurements”, is dedicated to other types of hardware which also generate data complementing the data generated by the MMP hardware (see Fig. 2.9). In other words, this tab opens the door to integrate other types of hardware to the system. At present, these additional measurements correspond to the ecg signal, the body impedance and the photoplethysmography.

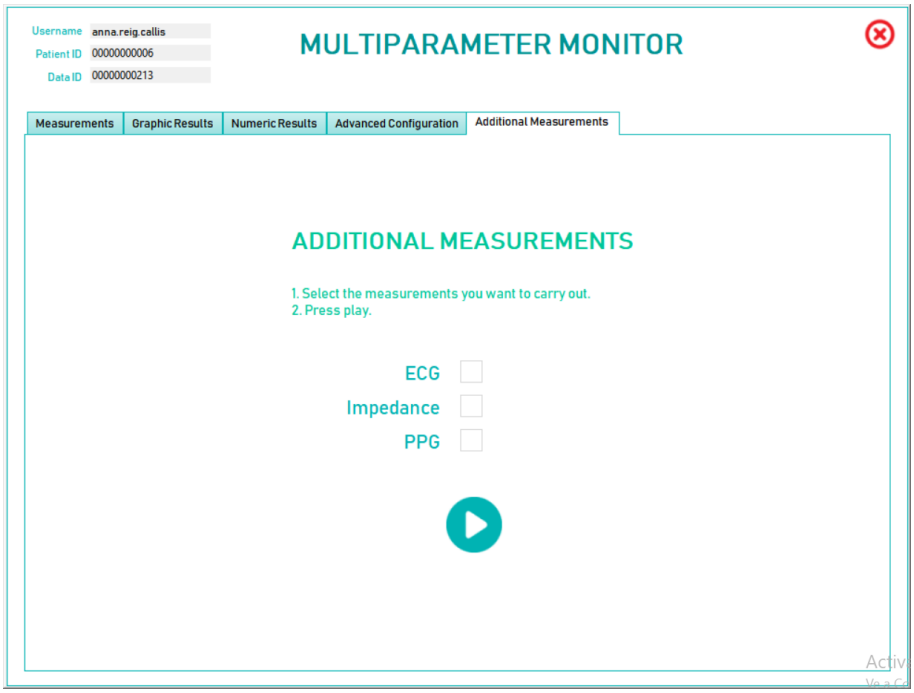


Figure 2.9: ”Additional Measurements” tab view

2.4.4. Doctor's view

If instead of logging in with a patient user name, a doctor user name is introduced, the behavior of the front-end will be completely different. Rather than taking measures, the front-end gives the possibility to the medical professional to replay a set of previous measurements of a certain patient. To do so, the professional must introduce the patient user name in which he or she is interested in and choose, from all the previous measurements done up to the moment, the desired one (see Fig. 2.10).

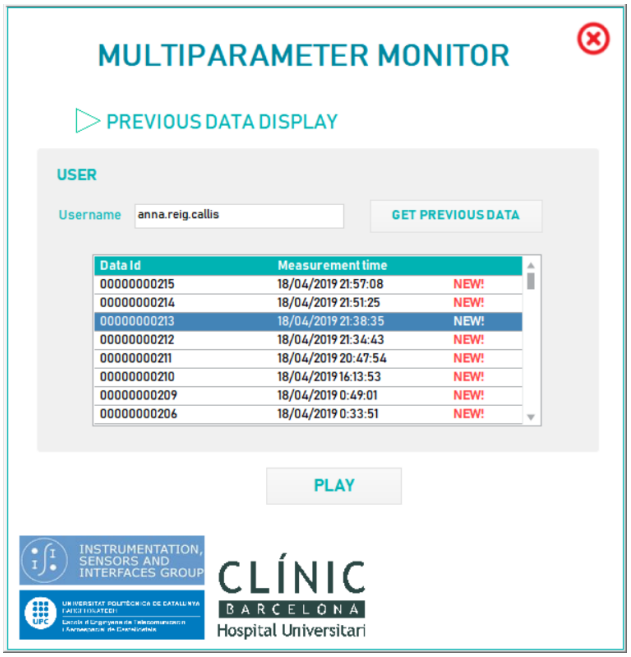


Figure 2.10: Selecting a set of previous measurements of a already known patient

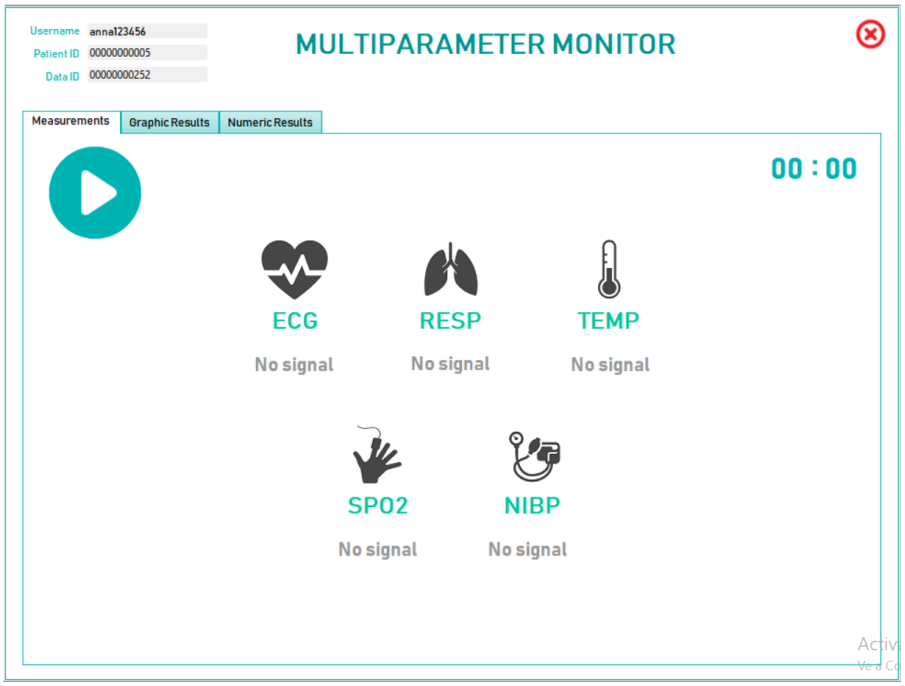
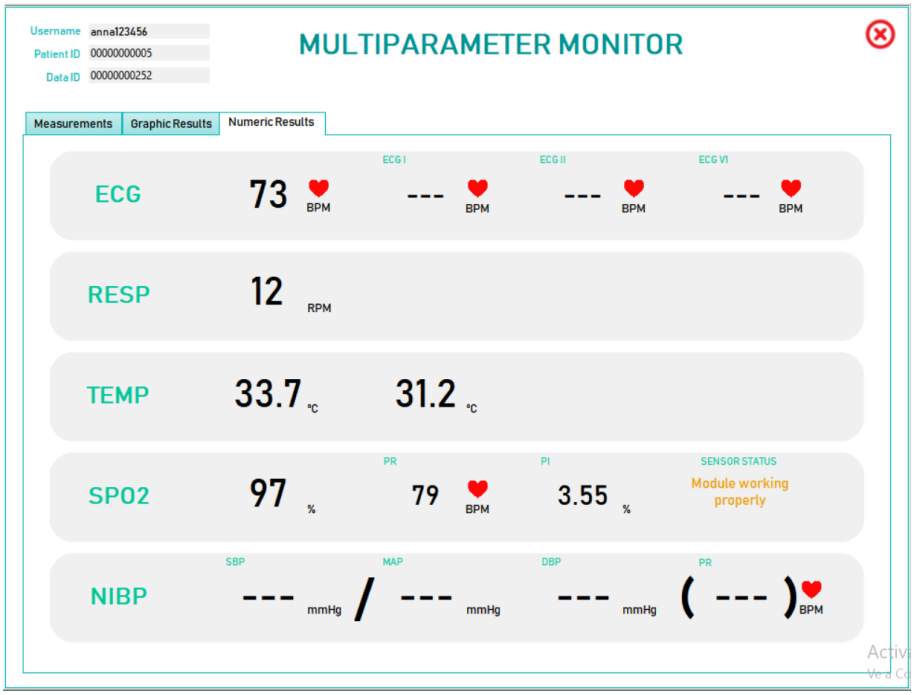
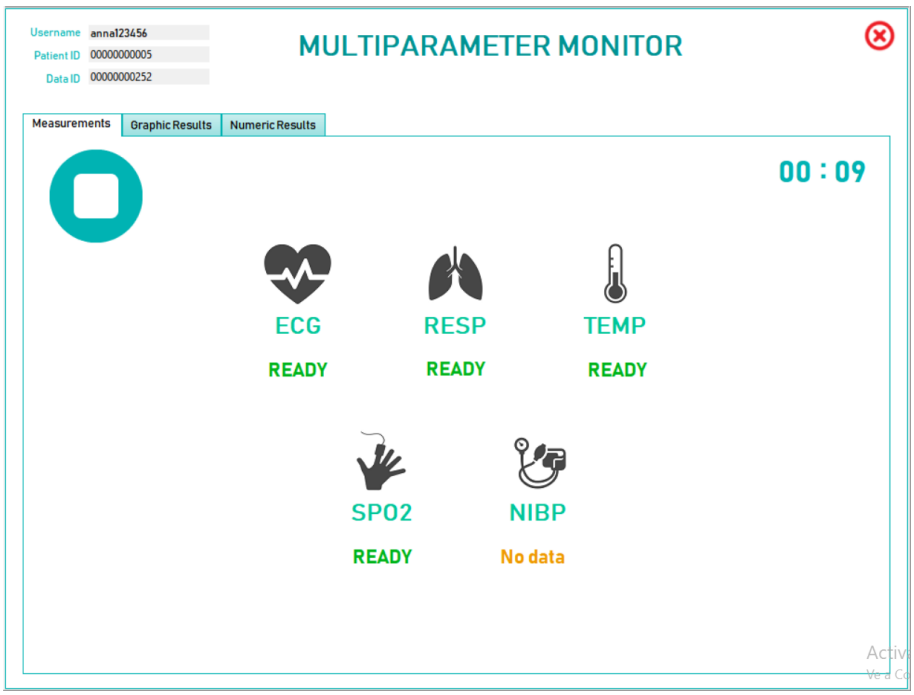


Figure 2.11: Front-end initial tab in its initial state

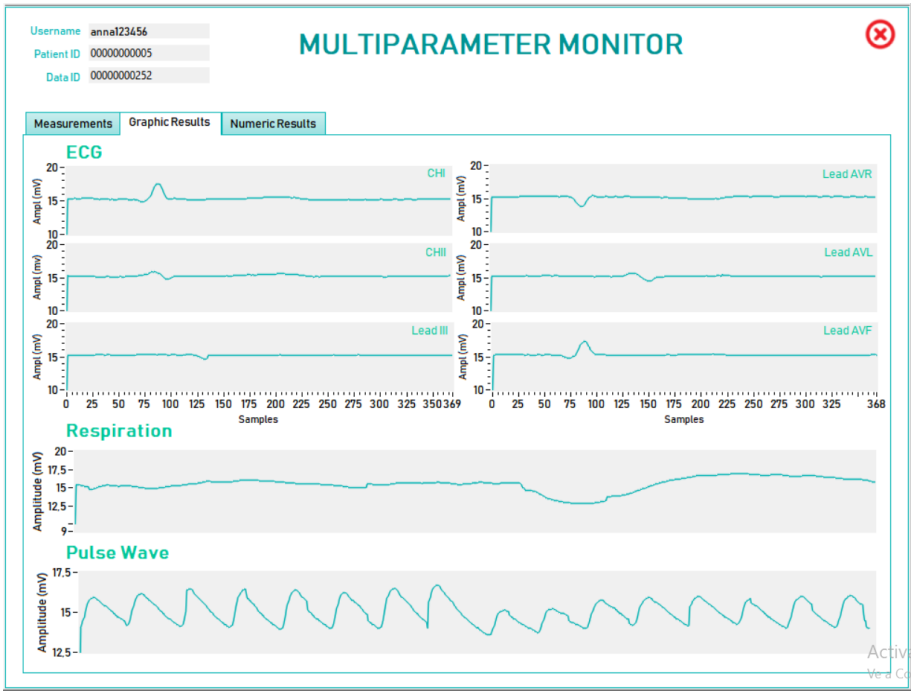
In this case, after successfully choosing the set of measurements to be replayed, a new window will appear similar to the one shown to the patient but instead of having five tabs, it consists of only the first three tabs with no required password in any of them (see figs. 2.11 and Fig. 2.12). By clicking on the play button, the measurements replay will start. In the same way, by clicking on the stop button, the replay will stop and reinitialize all the variables.



(a) ”Measurement” tab view



(b) ”Graphic Results” tab view



(c) "Numeric Results" tab view

Figure 2.12: Tabs of the doctor's view

2.5. Integration between the parts

After describing the different parts of the system independently, it is time to look at them as a set, allowing the whole system to work as expected.

Let's start by analyzing the relationship between the parts from Figure 2.13. As already stated in the Section 2.2., 2.4. and 2.3., the system is composed by three actors: the back-end, the front-end and the database, and each of them exchanges data with the two others.

To have a better understand of Figure 2.13, notice that the blue arrows represent the kind of data exchanged and thus, can be unidirectional or bidirectional depending on them. Furthermore, the bidirectional orange arrows indicate the library or communication method used between two elements.

Regarding the back-end, the communication with the front-end is based on TCP/IP sockets (whose theory is explained in Ref. [20]) and is made of two actions: sending the generated data by the MMP hardware through the back-end to the front-end, and sending the front-end configuration from the front-end itself to the back-end, allowing the latter to start measuring. This front-end configuration contains the database IP, database port, database schema name, user name, user password, patient id and data id. After receiving this setting the back-end can establish a connection with the database through the use of the *libmysql.dll* and *mysqlcppconn.dll*. This connection is required to exchange the measurement configuration that is specified in Table 2.3, 2.4, 2.5.

Concerning the complex relationship between the front-end and the database, as seen in Figure 2.13, it comprises two libraries which are *sql-LV.dll* and *sql_LV_oracle-mysql.dll* (see

Ref. [21] for the downloading source) and three actions. The first action, which is starts at the front-end, allows to save the configuration of the front-end and the configuration of the back-end measurements (both established by the user) to the database in such a way that the measurements configuration is accessible later for the back-end. Once the settings have been saved into the database, the second action, which is bidirectional, can take place. The second action consists of either sending measurements data from the front-end to the database in the case of having a patient user, or retrieving data from the database to the front-end in the case of having a doctor user. Note that the measurement data is not directly saved from the back-end aiming to speed up the communication process between this one and the front-end. The last action consists of retrieving the front-end configuration (i.e., the user name, data id, patient id) in case a doctor desires to view again a set of measurements.

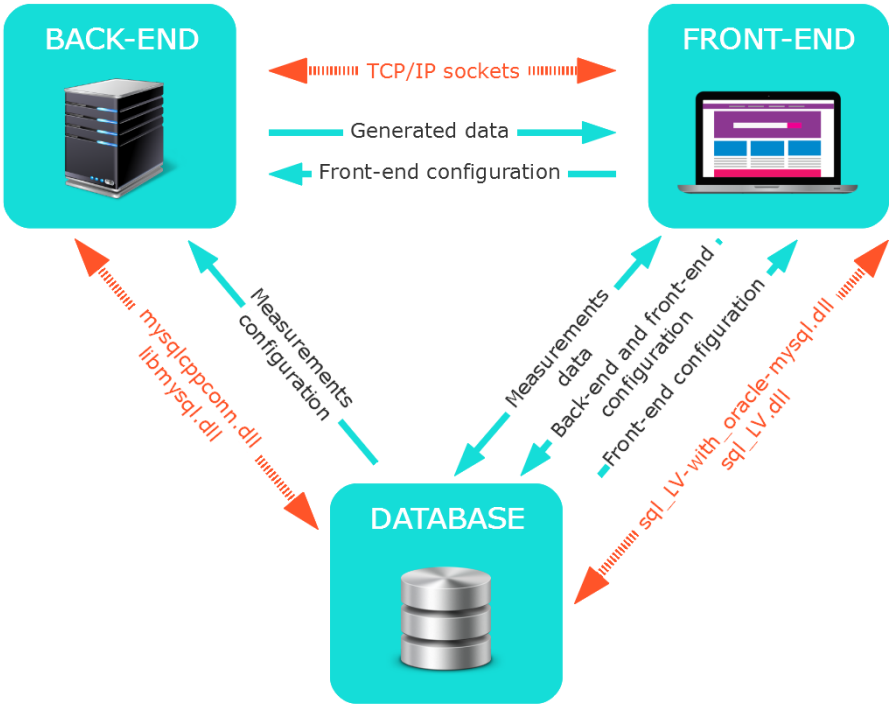


Figure 2.13: Relationship between the parts

CHAPTER 3. INTEGRATION OF THE MULTIPARAMETRIC SYSTEM INTO A PHYSICAL PLATFORM

This chapter describes the way in which the integration of the multiparametric system has been accomplished. In fact, the integration consists of three steps: a hardware integration, a software integration and a test with a real person to check that the system works as expected.

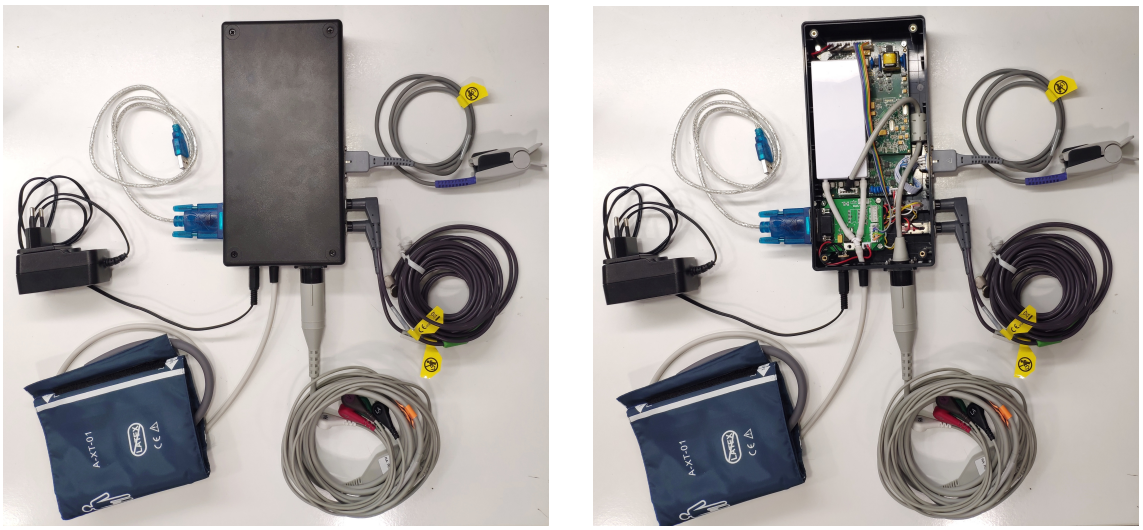
3.1. Main aim

The purpose of the integration is to obtain a first prototype of the previously designed system and hence, to demonstrate how portable it is. By proving the latter, the suitability of this system for developing countries as well as for heart condition homes is also demonstrated. Additionally, through this integration, the system is ready to be useful for society thereby fulfilling one of the main objectives of this project.

3.2. Hardware integration

The integration of the multiparametric hardware consists of two steps. Firstly, a box must be chosen to be the smallest one according to the hardware dimensions and must be provided with the corresponding holes to introduce the bulkhead unions that the hardware already has. Secondly, once the box perforated, the modules of communication and measurement must be closed into it (Fig. 3.1).

The box bought corresponds to the model named *ABS Hammond* 1599HBK and its dimensions are: 11 cm x 22 cm x 4.4 cm (see Ref. [22]).



(a) Hardware box closed

(b) Hardware box opened

Figure 3.1: Result of the hardware integration

3.3. Software integration

Similarly to the hardware, the software also requires to be integrated into a portable platform with its own Internet connection.

3.3.1. Mini portable computer

For this project, a mini portable computer such as *Microsoft* Surface Pro (Model 1514) (see Fig. 3.2 and Ref. [23]) has been the chosen platform to integrate the software. It is characterized by its Intel Core i5-3317 processor with 4GB memory and 128GB storage and its already pre-installed Windows8 Pro OS. The screen dimensions are 275 x 173 x 14 mm and the second-hand price of this PC is 209.77 $ (initially 350 €).

In this case, the previously developed software, that is the front-end and the back-end, has been installed into *Microsoft* Surface Pro (see Section A.1. for the proper installation). Concerning the database, it is possible to install it into the same PC or into another one. However, to secure the saved data, it is highly recommended to separated the database from the front-end and the back-end, which can be manipulated by the final user. In this case, as it is a prototype, the database has been installed into *Microsoft* Surface Pro as well.

Figure 3.2: *Microsoft* Surface Pro [23]

3.3.2. 3G/4G USB modem

A 3G/4G USB modem with a SIM card has been added to *Microsoft* Surface Pro, which will provide the PC with Internet access anywhere, and will allow to locate independently the database anywhere in the world as long as it is provided with Internet connection.

In this way, this modem gives the possibility to the system to be used in almost everywhere in the world, as long as there is 3G or 4G coverage, as Figure 3.3 shows. Notice that, for this map, it is implied that places with 4G already have 3G and in the same way, places with 3G already have GSM. In other words and due partially to this modem, the patient and the medical professionals do not need to be close to each other to carry out and analyze the physiological measurements anymore, thus giving rise to the concept of telemedicine.

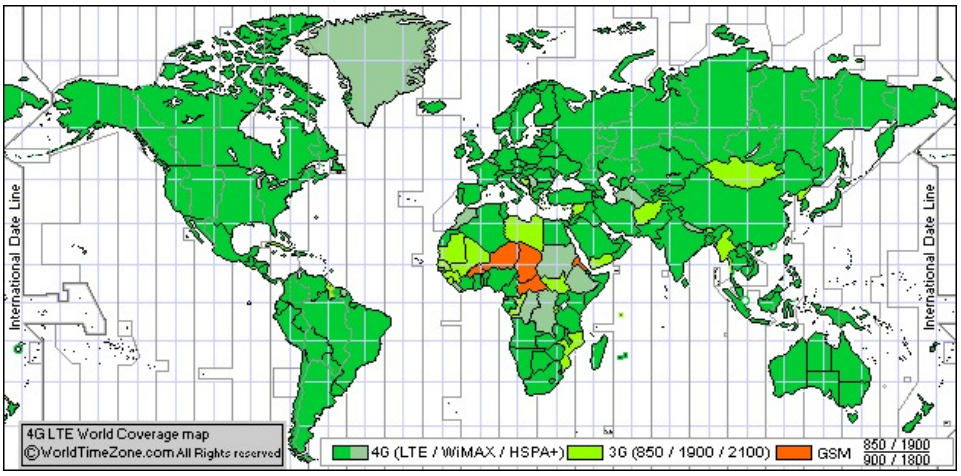


Figure 3.3: 4G LTE World Coverage Map [24]

This modem has been chosen among different options taking into account 3G and 4G band support, Windows 8 and 10 support, novelty and price. According to these features, Table 3.1 has been elaborated, from which it can be deduced that the two options fulfilling the features expected are *Huawei* E8372 and *Huawei* E3372. However, the first one is a little bit newer than the last one and the price difference is really small, that is why the most suitable modem is *Huawei* E8372 (see Fig. 3.4 and see Ref. [29]).

Model	3G	4G	Win8	Win10	Price
Docooler 4G [25]	X	X	X		27.99 €
D-Link DWM-222 [26]	X		X		72.98 €
Huawei E3372 [27]	X	X	X	X	43 €
Huawei E3531i-2 [28]	X				24.88 €
Huawei E8372 [29]	X	X	X	X	45.99 €

Table 3.1: Features of the selected 3G/4G USB modems

Figure 3.4: *Huawei* E8372 [29]

3.4. Test of the final integration

To check that the integration of the whole system works as expected, a test with a real patient must be performed. The aim is to demonstrate the system is able to, on one hand, measure and view main vital constants of a human-being and, on the other hand, save and retrieve these main vital constants.

3.4.1. Final setup

The result of assembling the software integration and the hardware integration can be seen in Figure 3.5, where all the already mentioned hardware elements appear (i.e. *Microsoft* Surface Pro, the 3G/4G modem with the SIM Card and the MMP hardware).

As Figure 3.5 shows, a USB hub has been used to connect the 3G/4G modem as well as the MMP hardware to the PC, as *Microsoft* Surface Pro has only one USB port.

Notice that for this demonstration, all the required software has been previously installed in the displayed PC as already explained in Section A.1..

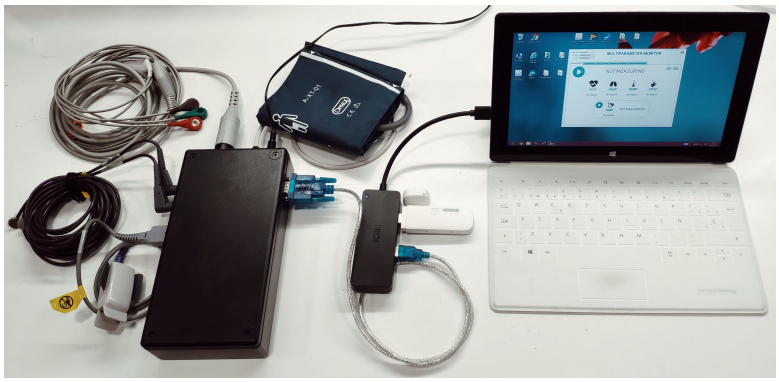


Figure 3.5: Elements integration result

3.4.2. Measurement part validation

To carry out the measurement part validation, real measurements have been taken with the director of this project who has been the patient to prove proper running of the system (see Figure 3.6).

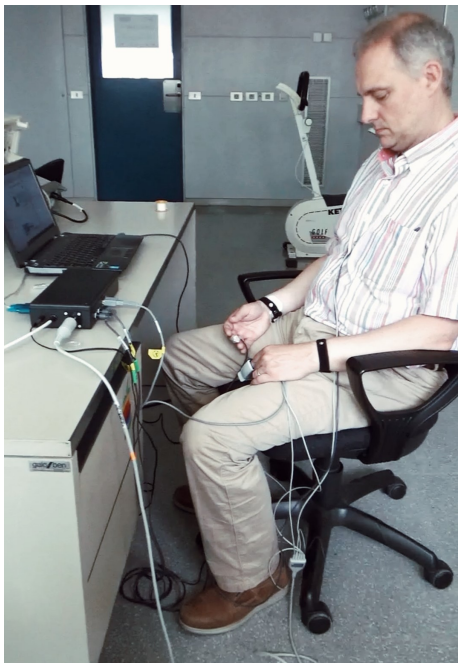
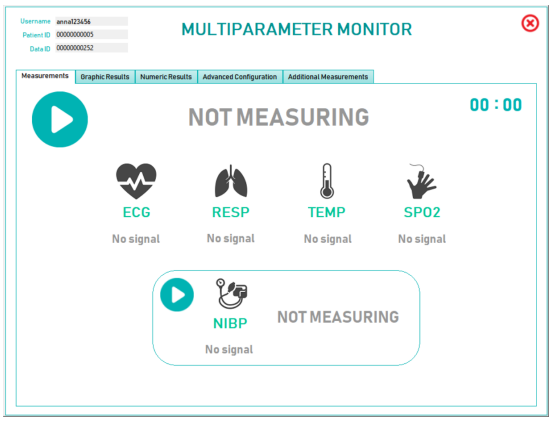


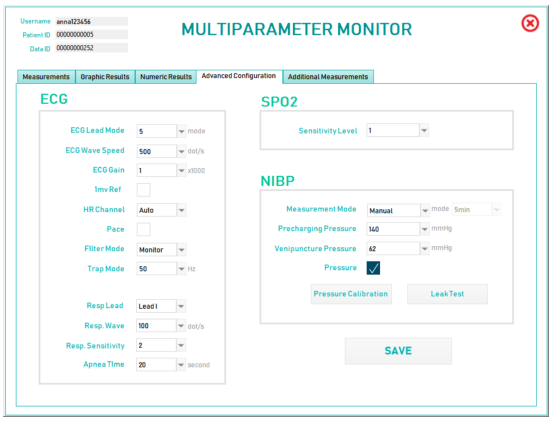
Figure 3.6: Taking real measurements with the director of this project

Before taking real measurements and after logging in with an already created user, such as in Section 2.4.3., the system has been reconfigured so to activate the filter of 50 Hz in order to cancel the interferences caused by mains. In other words, that is equivalent to change ”Trap Mode” from ”None” to ”50 Hz” as Figure 3.7(b) demonstrates it.

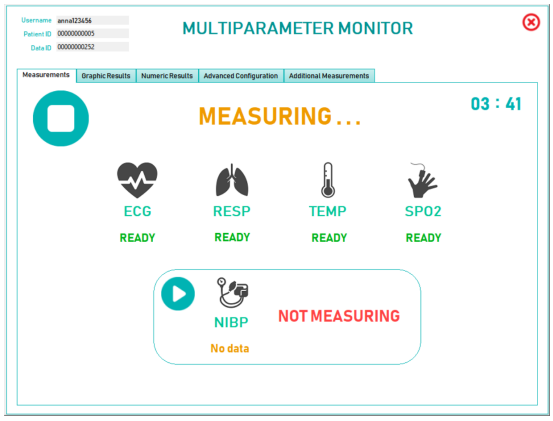
Once this setting applied, the measurements can start being recorded (see Fig. 3.7(c)).



(a) First screen before taking measures



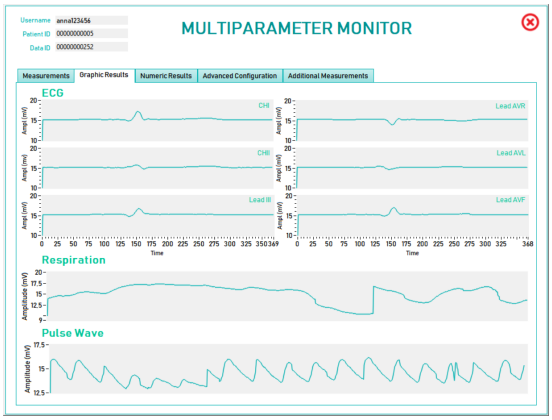
(b) Settings adjustment



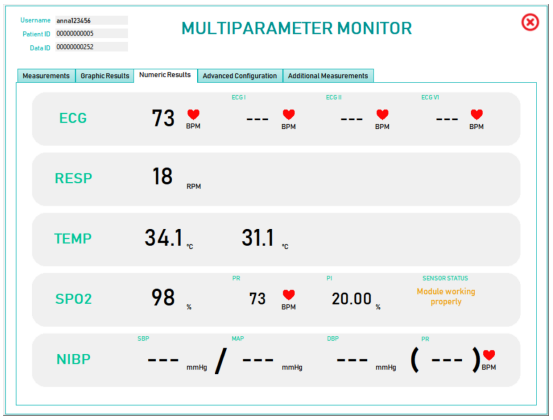
(c) First screen while taking measures

Figure 3.7: Screen shots before and while taking measures

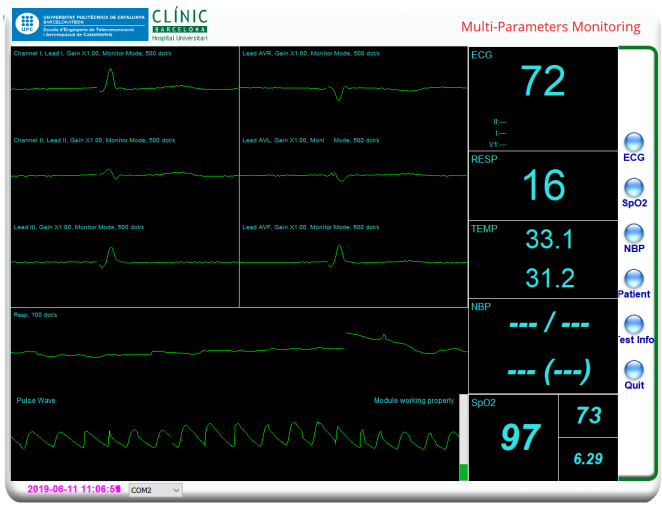
As exposed by Figure 3.8(a), 3.8(b) and 3.8(c), although the screen shots of the front-end and the back-end have not been taken in parallel at the same time, and thus, the observed values are not exactly the same, their relationship appear to be clear.



(a) Second screen



(b) Third screen



(c) Back-end screen

Figure 3.8: Comparison between the front-end data and the back-end data

Let's analyze the displayed parametric values in order to validate the functioning of the designed MMP system for measuring and viewing the principal vital constants of a human-being.

To accomplish it, Table 3.4.2. makes a comparison between the back-end values, the front-end values and the standard range of the variables measured. From this Table, it can be said that between the back-end values and the front-end values, there is not almost any difference and that all the measured parameters are within their standard range. In addition, ECG, RESP and SPO2 waves generated by MMP system, appearing in Figure 3.8(a), maintain the same shape as the ones generated by other parametric systems such as *Biopack* MP36. Through similarity, the shape of these waves is validated. Hence, this means that the patient is physically healthy and that the previously mentioned functions work as expected.

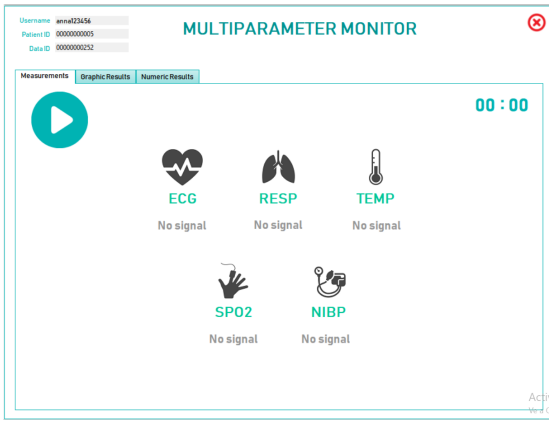
Notice that the PI is highly variable and that the measured temperature corresponds to the skin temperature, that is why the displayed values are lower than standard body temperature. Moreover, the high similarity between ECG HR value and SPO2 PR value validates the good functioning of SPO2. The difference between these two previous variables is due to 1% tolerance of ECG HR and 2% tolerance of SPO2 PR.

	Back-end	**Front-end**	**Standard range**	**Validation**
HR [bpm]	72	73	60-100 [30]	OK
RESP [rpm]	16	18	12-20 [30]	OK
TEMP [℃]	33.1 / 31.2	34.1 / 31.1	31-35 [31]	OK
SPO2 [%]	97	98	95-100 [32]	OK
PR [bpm]	73	73	60-80 [30]	OK
PI [%]	6.29	20.00	0.02-20 [33]	OK

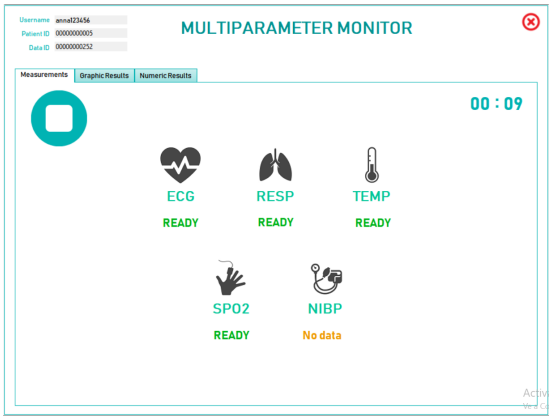
Table 3.2: Validation and comparison between back-end values, front-end values and standard ranges

3.4.3. Replay part validation

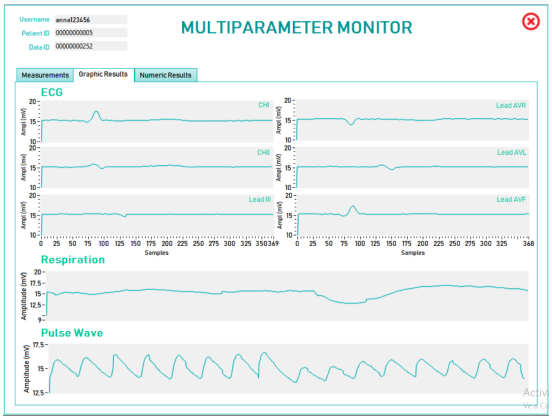
Concerning the measurements replaying part (see Figure 3.7(a), 3.7(c), 3.8(a) and 3.8(b)), the observed values in the replay screen shots (i.e. Figure 3.9(c) and 3.9(d)) are not exactly the same as the values of the previous screen shots (i.e. Figure 3.8(a) and 3.8(b)), as they have not been taken at the same instant. However, the relationship between the parametric values seen while taking measurements and the parametric values displayed while replaying these measurements appear to be clear.



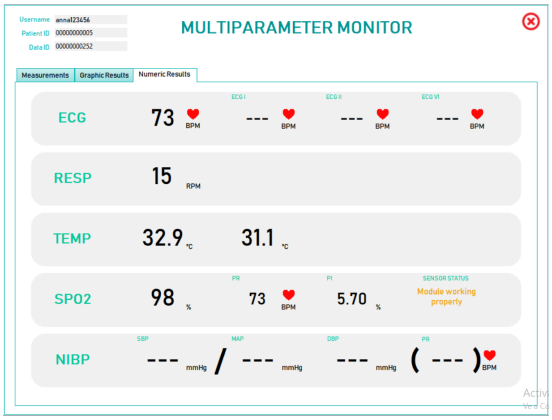
(a) Second screen



(b) Third screen



(c) Back-end screen



(d) Back-end screen

Figure 3.9: Screen shots while replaying the already taken measurements

Let's analyze the parametric values of the measurements replaying part in order to validate the functioning of the designed MMP system for saving and replaying the principal vital constants of a human-being.

To accomplish it, Table 3.4.3. makes a comparison between the front-end values while taking measures, the front-end values while replaying them and their standard ranges.

From this Table, it can be said that the replayed and measured parameters are almost the same; the variation is due to the moment in which the screen shot was taken. Furthermore, all the replayed parameters are within their standard range. In addition, ECG, RESP and SPO2 waves replayed, appearing in Figure 3.9(c), maintain the same shape as the ones generated by MMP system. Through similarity, the shape of these waves is validated and thus, the previously mentioned functions work as expected.

Notice that through the use of ”DataId” parameter, it is possible to check that the replayed measurements are the ones corresponding to the taken ones. In this case, for both parts, ”DataId” is equal to 252.

	Measurement	**Replay**	**Normal range**	**Validation**
HR [bpm]	73	73	60-100 [30]	OK
RESP [rpm]	18	15	12-20 [30]	OK
TEMP [℃]	34.1 / 31.1	32.9 / 31.1	32-35 [31]	OK
SPO2 [%]	98	98	95-100 [32]	OK
PR [bpm]	73	73	60-80 [30]	OK
PI [%]	20.00	5.70	0.02 - 20 [33]	OK

Table 3.3: Validation and comparison between the measurement part values, the replay part values and standard ranges

CHAPTER 4. CONVOLUTIONAL NEURAL NETWORK FOR THEORETICAL DETECTION OF MOSQUITOES AND THEIR HABITAT

After prototyping a system able to detect diseases transmitted by mosquitoes in a human being, such as malaria, the following chapters are going to be focus on how to prevent this deadly disease by detecting the direct cause: the mosquitoes and their larvae. In other words, this implies detecting mosquitoes and puddles (where larvae generally grow) from a picture by using Artificial Intelligence (AI) and the complementary help of ground and air sensors data. Then, once the detection done, it is about either preventing the closest population about the risk in the zone or applying the corresponding extermination methods.

AI is probably the most revolutionary technology of the last decades and applied to computer vision, can help to recognize objects, persons, and animals in images and videos. To accomplish that, a neural network composed by a set of layers and neurons is ”trained” by feeding it with thousands of examples, from which it will learn the most useful patterns to detect the final target (see Ref. [34]). In this project, deep learning for computer vision will be applied to adjust a pre-trained convolutional neural network (CNN) called *Inception* to be able to detect mosquitoes and their habitat in images.

The structure of this chapter can be divided in two: on one hand, *Inception* model as well as the network data sets are defined; and on the other hand, according to the generated data sets, it is described how the convolutional neural network is trained and validated.

4.1. Aim

Through the use of CNNs two goals are pursued. The first one is to detect if there is a puddle or stagnant water in a picture, in such a way that the locations of habitats where mosquitoes larvae can potentially grow, are detected. The second aim consists of detecting the presence of a mosquito in a picture and of classifying it between different considered species in order to further on, determine the density of mosquitoes of each class in a picture. This will allow, for instance, to estimate the potential risk of transmission of diseases carried by mosquitoes according to the environmental conditions, or to apply a mosquitoes extermination method in case these insects are located near populated zones.

The idea is to have two trained and validated CNNs: one for puddles detection and the other for mosquitoes detection and classification according to the kinds of mosquitoes with which the network was trained.

4.2. *Inception* v3 model

As stated by the references [35] and [36], Inception v3, designed by *Google*, is a widely-used image recognition model (based on reference [37]) and the third version in a series of Deep Learning Convolutional Architectures. In fact, this model was built by using *TensorFlow*, a framework constituted by *Google* for creating deep learning models.

According to reference [38], this CNN has been trained on more than a million images from the *Google ImageNet* database. Moreover, it is 48 layers deep and can classify images into 1000 object categories such as keyboard, mouse, pencil, and many animals. As a result, the pre-trained network has been shown to attain greater than 78.1% accuracy on the *ImageNet* training data set(see Ref. [35]).

The model itself consists of two parts (see Ref. [39] and Fig. 4.1): the first one corresponds to the general feature extraction part with a convolutional neural network (see Ref. [40] to have a deeper understanding), while the second one is a classification part based in the features of the first part with fully-connected and softmax layers (see Ref. [41] about softmax). Therefore, when a new model is built to classify the original data set (which at least must have 20 pictures), the feature extraction part is reused and the classification part with the data set retrained. Since it is not needed to train the feature extraction part (the most complex part), the model can be trained with less computational resources and time. This feature allows to easily retrain the network as many times as desired and hence, it makes the model very suitable for mosquitoes and puddle detection.

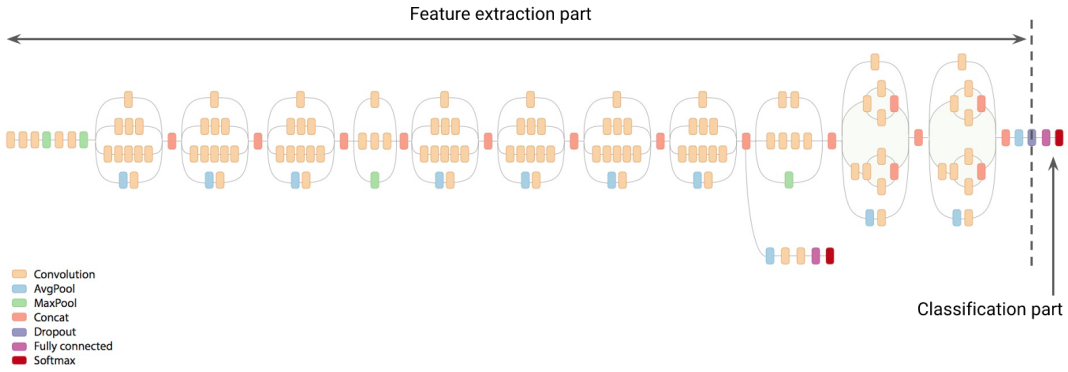


Figure 4.1: High-level diagram of the *Inception* v3 model [39]

4.3. Network data sets

Through the use of *Inception*, two new models are going to be built as already explained in Section 4.1.. Building two new models mean having two new data set, that is having a data set for classifying puddles and a data set for classifying mosquitoes. Each data set has a certain number of classes, among which the network classifies any inputted picture.

4.3.1. Puddles data set

Concerning the puddles data set, two classes are defined: ”puddle” and ”miscellaneous”. The first class contains 120 different pictures of puddles, stagnant waters, aquatic habitats suitable for mosquitoes larvae growth, either next to natural, agricultural, urban or semi-urban areas. The second class consists also of 120 pictures of areas that can be easily found next to stagnant water zones but without any water trace on them. To have an idea of the pictures included in these classes, Figure 4.2 and 4.3 contains some samples.

Regarding these samples, in the same column have been grouped the same kind of picture with and without water in different areas. In particular, the first and the last column

correspond to natural and agricultural areas, while the second and the third are associated to urban and semi-urban areas. In this way, by comparing the same type of pictures with and without water, the algorithm shall learn how a puddle or stagnant water is.

Figure 4.2: Picture samples of the ”puddle” class

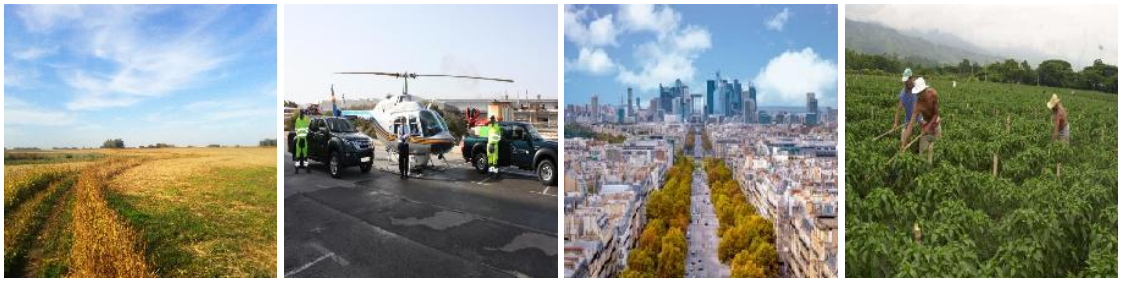


Figure 4.3: Picture samples of the ”miscellaneous” class

4.3.2. Mosquitoes dataset

Four classes are defined in the mosquitoes data set: ”aedes albopictus”, ”anopheles atroparvus”, ”culex pipiens” and ”miscellaneous”. The first three classes correspond to the main three species of mosquitoes found in Spain according to the reference [42] and contain 86 pictures of each specie separately. The last class has also 86 pictures, but they are about places where mosquitoes can be found such as fingers, hands, leaves or stagnant waters. Figure 4.4, 4.5, 4.6 and 4.7 give an idea of what is included in these classes.

Concerning the next samples shown in Figure 4.4, 4.5, 4.6 and 4.7, in the same column have been grouped the same kind of picture with and without mosquito in different backgrounds. In particular, the first column correspond to a leafy background, the second to a whitish background, the third to a background with a finger and the last to a flowered background. It should be noted that for this last column, the background of the ”anopheles atroparvus” class does not contain flowers but a trunk as no flowered background was found for this case. In this way, by comparing the same type of pictures with and without mosquito, the algorithm shall learn when there is a mosquito and to what specie it belongs.

As it is desired to validate this network later on with pictures of mosquitoes taken with a real camera, mosquitoes found in Spain have been chosen to elaborate this data set (as the pictures will be taken in Castelldefels, Barcelona). Therefore, as it is obvious, the type of mosquitoes present in the dataset must vary depending on the mosquitoes present in the region where the pictures are taken.

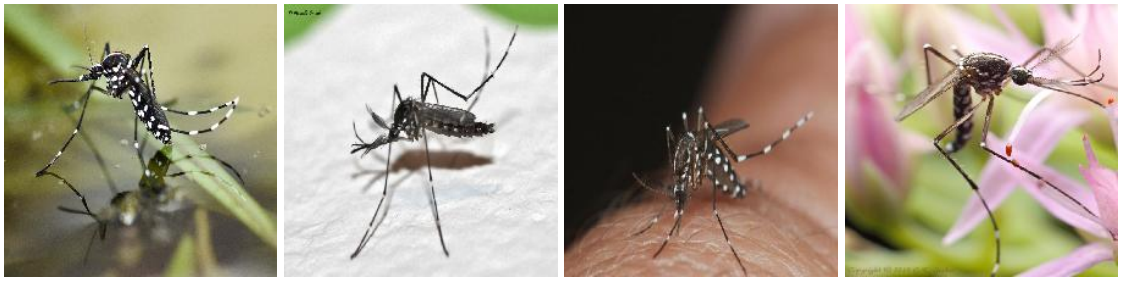


Figure 4.4: Picture samples of the ”aedes albopictus” class

Figure 4.5: Picture samples of the ”anopheles atroparvus” class

Figure 4.6: Picture samples of the ”culex pipiens” class

Figure 4.7: Picture samples of the ”miscellaneous” class

4.3.3. Considerations

Related with the data sets used, three considerations must be taken into account:

- All the pictures contained in each class have been downloaded from *Google Images* and carefully selected.

- All the classes contain the same number of pictures in order to avoid giving more weight to a class than to another and thereby, to avoid influencing the final results.
- The number of pictures found on the Internet about the evaluated types of mosquitoes is quite scarce which could affect to the probabilities of well-classifying a mosquito. In case of having a larger database with pictures of the studied mosquitoes, the probabilities of correctly classifying a mosquito would increase significantly.

4.4. Network training

The way of training the two networks derived from the already mentioned sets of images is almost the same. In this case, and as already described in Section 4.2., the network training consists of retraining *Inception*'s final layer for the specific images of the dataset and their corresponding categories, which were assigned manually prior the training phase. Fortunately, this can be accomplished by simply running a very convenient script developed by *Google*'s *Tensorflow* team.

4.4.1. Training theory

This script, called "retrain.py" (from *TensorFlow*, see Ref. [46]), loads the pre-trained module and trains a new classifier on top for the pictures of the defined data set and the associated labels, or categories.

4.4.1.1. Method used

According to the reference [43], the method used for retraining *Inception* model can be divided into some phases.

The first phase analyzes all the images on disk and calculates and caches the bottleneck values for each image of the training set. "Bottleneck" defines the layer just before the final output layer that actually does the classification. This penultimate layer has been trained to output a set of values that is good enough for the classifier to use to distinguish between all the classes it's been asked to recognize.

In a second phase, the script splits them into stable training, testing, and validation sets which by default these sets contain 80%, 10% and 10% of the original pictures, respectively. Note that the algorithm will be trained on the training set, and as such it will not see the images of the test set until the very last end of the process, when the accuracy of the algorithms on images that has not seen before is assessed.

Once the bottlenecks and the batches are complete, the third phase and thereby, the top layer training begins. By default this script will run 4,000 training steps. Each step chooses a mini-batch of 10 images selected at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels to update the final layer's weights through the back-propagation process. As the process continues the reported accuracy in the training set should improve.

After all the steps are done, a final test accuracy evaluation is run on the images of the testing set. Indeed, this test evaluation is the best estimate of how the trained model will perform on the classification task.

4.4.1.2. Inputs of the training script

Before running the script, 7 mandatory inputs must be set although they are not the only inputs the script has. Other optional arguments can be specified to fine-tune the training process according to the user's needs.

First, the bottleneck directory must be indicated using the variable "bottleneck_dir", that is the directory where all the generated files related to bottlenecks are going to be saved. In the same way, the location of directories of *Inception* model and the data set must be also inputted through the variables "model_dir" and "image_dir", respectively.

Although by default the number of training steps corresponds to 4000, it must be specified through the variable "how_many_training_steps". Indeed, after a certain number of steps, the algorithm converges into a result that do not improve increasing the number of steps. However, the training time increases linearly with the number of steps.

Moreover, the script will write out the new model trained (i.e. the weights of the connections between neurons) to the file specified by the variable "output_graph" and a text file containing the labels to one specified by the variable "output_labels". In the same way, the script will log algorithm summaries to the directory indicated by the variable "summaries_dir". If it is desired to classify new images and the network has been already trained, there is no need to retrain it again, as the model can be recovered from these output files.

4.4.1.3. Outputs of the training script

As stated by reference [43], a series of step outputs are displayed on the screen while the training is executed, each one showing training accuracy, validation accuracy, and the cross entropy.

The training accuracy shows what percent of the images used in the current training batch were labeled with the correct class. In fact, it is based on images that the network has been able to learn from, so the network can over-fit the training data (see Ref. [34]).

The validation accuracy is the precision on a randomly-selected group of images from the validation set. If the training accuracy is high but the validation accuracy remains low, that means the network is over-fitting and memorizing particular features in the training images, not generalizing correctly, which means that has difficulties to classify with accuracy images that has not seen before.

Cross entropy is a loss function which gives a glimpse into how well the learning process is progressing (see Ref. [44]). The goal of the training process is to find the best weights such that a given cost function is minimized. The cross-entropy is the cost function to be minimized when training*Inception* model in a new data set. By observing the loss trend, it is possible to know if the learning is working or not.

Finally, after the model is fully trained, the final test accuracy is computed based on the percent of the images in the testing set that are given the correct label. Remember that none of these images were used to train the algorithm.

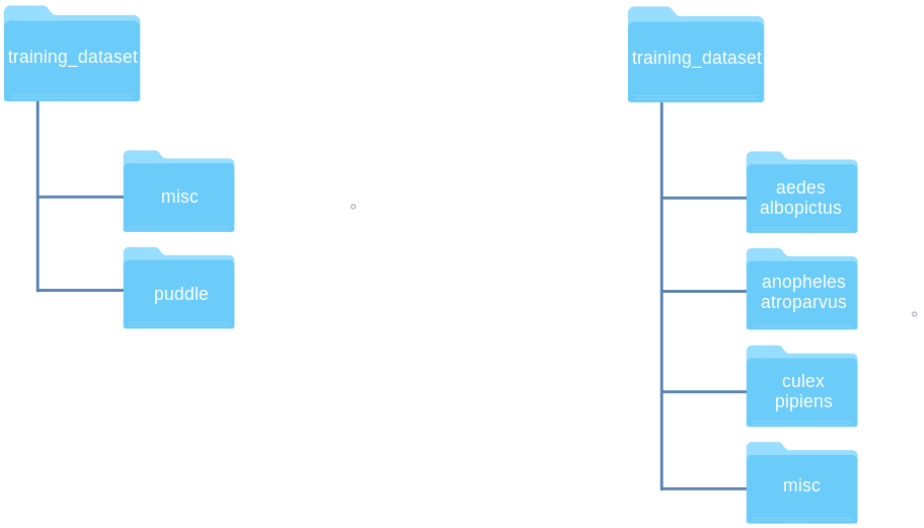
4.4.2. Network setup

To retrain *Inception*'s final layer, first, the next github repository must be cloned in order to download *Inception* model.

```
git clone https://github.com/koflerm/tensorflow-image-classifier.git
```

Then, just as explained by the reference [45], to tell *Inception* the correct label for each picture, some steps must be carried out. First, a folder called "training_dataset" must be created into the same cloned folder. (If it is desired to create the two new models into the same cloned folder, consider calling differently the folder where the data set is saved, that is "training_dataset_puddle" and "training_dataset_mosquitoes"). Immediately after, this new folder must be filled with a number of sub-folders equal to the different classes that the data set has to classify. Finally, these subfolders must be named keeping in mind that these names will be associated to the labels *Inception* will use.

In other words, for the puddle data set and the mosquitoes data set, the folders structures are the ones shown in Figure 4.8(a) and 4.8(b), respectively.



(a) Puddle folders structures

(b) Mosquitoes folders structure

Figure 4.8: Folders structure

Next, it is time to download pictures from *Google Images* in order to fill these sub-folders according to the established classes. Since downloading every single image manually would be very time-consuming, a *Chrome* extension named *Fatkun Batch Download Image* has been used. This extension allows to automatically download images from *Google* with minimum effort from the user's point of view. Note that after downloading and selecting carefully the images, they must be strictly converted to JPEG format.

4.4.3. Network execution

After setting up all directories and data sets, the train can begin. The next command must be executed in the *Python* shell. The executed script installs the *Inception* model (if not already installed) and initiates the retraining process for the specified image data sets.

The bottleneck files, the training summary, the retrained graphs and the retrained labels will be saved in a folder named ”tf_files”.

```
python retrain.py --bottleneck_dir=tf_files/bottlenecks --how_many_training_steps=2000
    --model_dir=inception --summaries_dir=tf_files/training_summaries/basic --
    output_graph=tf_files/retrained_graph.pb --output_labels=tf_files/retrained_labels.
    txt --image_dir=training_dataset
```

Once the process is complete, it returns a final test accuracy which for the puddle network corresponds to 94.1% and for the mosquitoe network to 79.5% (see Fig. 4.9 and 4.10). This means that 94.1% of the images in the puddle testing set and 79.5% of the images in the mosquitoes testing set, are given the correct label.

```
2019-05-07 10:49:45.089762: Step 1999: Train accuracy = 100.0%
2019-05-07 10:49:45.090260: Step 1999: Cross entropy = 0.011534
2019-05-07 10:49:45.338299: Step 1999: Validation accuracy = 95.0% (N=100)
Final test accuracy = 94.1% (N=34)
```

Figure 4.9: Final test accuracy for puddle network

```
2019-05-03 18:21:01.638145: Step 1999: Train accuracy = 100.0%
2019-05-03 18:21:01.638145: Step 1999: Cross entropy = 0.078773
2019-05-03 18:21:01.898164: Step 1999: Validation accuracy = 84.0% (N=100)
Final test accuracy = 79.5% (N=78)
```

Figure 4.10: Final test accuracy for mosquitoes network

4.5. Network validation

After re-training the model, it’s now time to test it on pictures that have not been used for the training process. Note that this Section has been included just for illustrative purposes, since the accuracy in the test set already is a measure of the model’s performance on images not seen during training.

4.5.1. Validation method

To achieve it, the same number of pictures must be downloaded of each existing class in the data set. Then, these pictures must be saved all together in a new folder located into the cloned folder (that is the root of the project downloaded previously from *GitHub*).

Afterwards, the script ”theoretical_process.py” (see Section D for more details) must be executed in the *Python* Shell as shown below.

```
python theoretical_process.py classify.py -i="images_test" -o="output"
```

This script calls the script ”classify.py” (from *TensorFlow*) which classifies each picture found in the inputted folder ”images_test” using the trained network. For each one of the images located in the ”images_test” folder, the algorithm calculates the probability of belonging to each one of the possible classes. Note that these probabilities must sum 1. In addition, the class with higher probability with be the one finally assigned to that image. However, the robustness of the prediction (or reliability) must be also taken into account. For instance, when predicting the class of an image which has three possible classes, an

algorithm giving 98%/1%/1% probability to the three classes is preferable to one giving 60%/20%/20%, even if the same class prediction will be associated to the image. In fact, these probabilities are collected into a output file as well as their interpretation.

4.5.2. Validation for detecting puddles

For validating puddles detecting, a new set of pictures has been created with 10 pictures of puddles and 10 pictures of puddles environments but without any puddle on them.

After running the script ”theoretical_process.py”, the results show that 2 pictures out of 20 have been wrongly classified taking into account that the condition of determining a label consists of assigning the one having the highest probability. In other words, for this set of pictures, 90% of them have been well-classified which validates the trained model as its final test accuracy (94.1%) is really close to the obtained accuracy with new pictures. However, including a large number of samples, the 90% would converge to the 94.1% obtained by using a large test set.

A sample of these pictures and their probabilities are shown in Figure 4.12 and 4.11. As Figure 4.12 exposes, the algorithm has no doubt about the labels of the pictures called ”puddle6.jpg” and ”misc2.jpg”. However, although ”puddle8.jpg” has high chances of being a puddle, the algoritm is not 99% sure about it. For the case of ”misc10.jpg”, there are almost 50% of chances of containing a puddle or not but as the probability of containing a puddle is slightly higher than the one of not having a puddle, the picture is classified as having a puddle, although actually there is not.

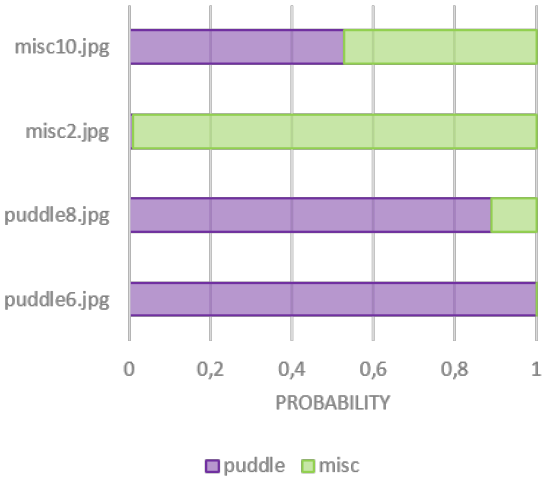
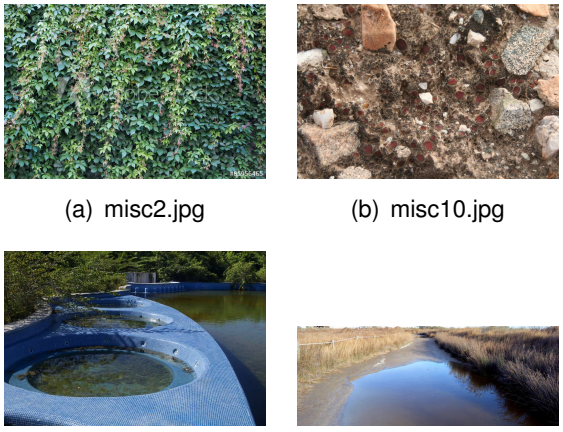


Figure 4.11: Probabilities distribution



(a) misc2.jpg (b) misc10.jpg

(c) puddle6.jpg (d) puddle8.jpg

Figure 4.12: Selected pictures

4.5.3. Validation for distinguishing one mosquito per picture

For validating mosquitoes classification, a new set of pictures has been created with 10 pictures of each type of studied mosquito and 10 pictures of mosquitoes environments.

After running the script ”theoretical_process.py”, the results show that 4 pictures out of 40 have been wrongly classified applying the same condition as Section 4.5.2.. In other

words, for this set of pictures, 90% of them have been well-classified which validates the trained model, as its final test accuracy (79.5%) is quite close to the obtained accuracy with new pictures. In fact, the obtained accuracy is even better than the final test accuracy. However, including a large number of samples, the 90% would converge to the 79.5% obtained by using a large test set.

A sample of these pictures and their probabilities are shown in Figure 4.13 and 4.14. As Figure 4.13 exposes, the algorithm has no doubt about the labels of the pictures called ”aedes11.jpeg” and ”misc1.jpeg”, and has almost no doubt about ”anopheles7.jpeg” and ”culex13.jpeg”. However, concerning the pictures named ”aedes3.jpeg” and ”misc5.jpeg”, although the label determination is correct, the algorithm is not 99% sure of their content. Finally, for ”anopheles10.jpeg” and ”culex2.jpeg”, the algorithm fails by confusing these two kinds of mosquitoes. In fact, just by precisely observing these pictures, it is even difficult for a human-being to distinguish between these two species as, according to reference [47], what differs between them is the length of some of their antennas, the body size and the body position above a surface.

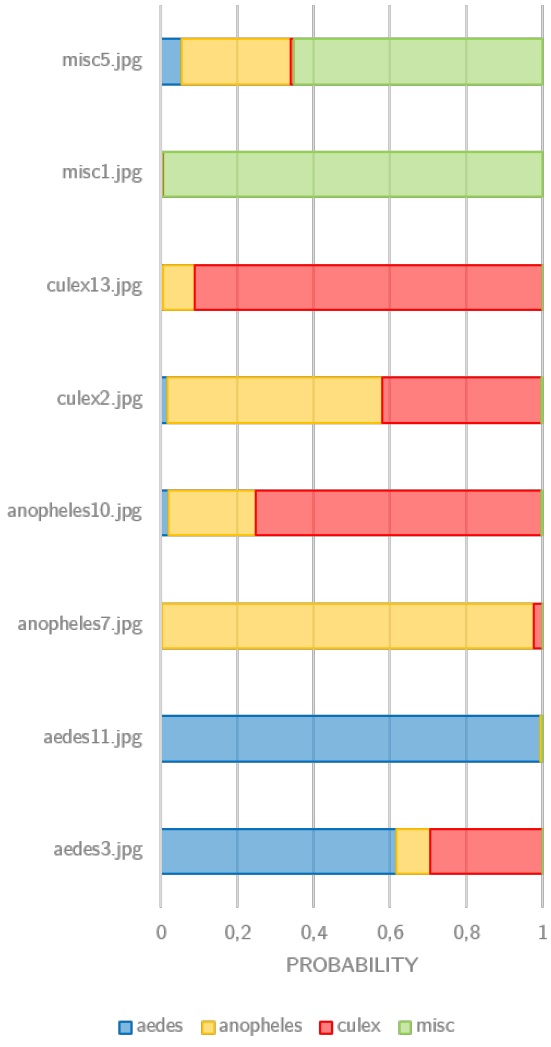
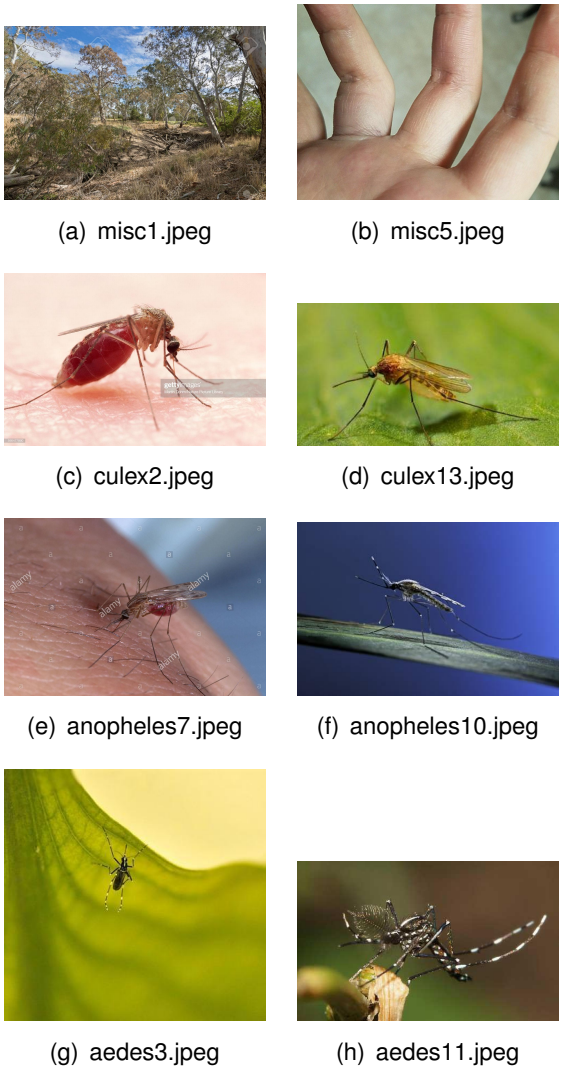


Figure 4.13: Probabilities distribution



(a) misc1.jpeg (b) misc5.jpeg

(c) culex2.jpeg (d) culex13.jpeg

(e) anopheles7.jpeg (f) anopheles10.jpeg

(g) aedes3.jpeg (h) aedes11.jpeg

Figure 4.14: Selected pictures

CHAPTER 5. DESIGN OF THE MOSQUITO PREVENTION SYSTEM

After demonstrating theoretically that, through the use of CNNs, it is possible to detect puddles and to classify mosquitoes from a picture, it is time now to apply this knowledge to fight against mosquito transmitted diseases.

In other words, in this chapter, the designed system to prevent mosquito transmitted diseases is going to be presented and described. This is going to lead to the quantification of the environmental conditions for larval growth and mosquito development, a fundamental requirement for such system. Then, the ground and air sensors architecture is going to be detailed. Finally, the choice of a camera able to take pictures similar to the ones downloaded from the Internet, like in Section 4, is going to be carried out.

5.1. Aim

What it is intended with this chapter is to design a system combining AI and the quantification of environmental conditions to successfully detect the presence of mosquitoes and their larvae.

In fact, the probability of successfully identifying mosquitoes and puddles from pictures taken in a given area, increases noticeably when combining AI and the quantification of environmental conditions than when only using AI. Moreover, quantifying environmental conditions simply means gathering data of ground and air sensors so to be able to know if the necessary conditions for the development of mosquitoes and their larvae are fulfilled.

Therefore, the fight against mosquitoes near populated zones can be implemented in three phases.

The first phase is focused on mosquito larvae. In fact, if larvae are located and eradicated, mosquitoes can not born and breed again. Hence, this phase consists of, determining, through the use of water local sensors and air sensors, if the conditions for larvae grow are fulfilled. Then, if it is the case, it is about locating stagnant waters by taking pictures with a camera fitted in a drone or rover. Nevertheless, once puddles have been located, taking advantage that the land orography does not generally change abruptly, the puddle model can be extracted after studying a specific puddle of the studied area which makes the area scanning not necessary anymore. As soon as the zones with puddles potentially infected with larvae, are known, any eradication larval method can be applied.

Once the first phase accomplished, the second phase can be executed. One could think that larvae elimination is enough for finishing with mosquitoes in populated areas. However, only puddles situated in public areas can be controlled, that means that the ones in private areas fulfilling the conditions for larval growth, can give rise to the birth of new mosquitoes not initially expected. Once born, these mosquitoes fly anywhere and that is why they must be also detected. The detection is carried out determining, through the use of air sensors, if the mosquito development conditions are fulfilled. If it is the case, a camera fitted in a drone or a rover, takes pictures of the area. From these taken pictures, the mosquito density in a specific location can be determined.

Finally, by knowing the puddles location and their state as well as the mosquito density and

the mosquito specie present in a studied area, it can be decided whether the third phase should be applied or not. This last phase is about taking action such as, for instance, preventing nearest population close to the studied zone of the existing risk, releasing sterile male mosquitoes in the studied area to avoid the propagation of mosquitoes or, even applying a certain fumigation mode.

To sum up, through this multi-information system, it is possible to optimize the results given by AI algorithm, avoiding to sweep constantly the area to be controlled.

5.2. System definition

The system has been designed for being able to detect the presence of mosquitoes and their larvae in a given area.

To accomplish it, for the case of larval detection, pictures of the studied area are taken with a camera. Immediately after taking each picture, the CNN, trained for puddle detection, classifies it, thus, obtaining for each picture, the probability of containing a puddle and the probability of not containing it. Moreover, asynchronously, water environmental data are gathered. Once pictures have been processed by the CNN, by a joint analysis of these water environmental data and the processed pictures, the probability of having detected puddles with larvae is determined.

For the case of mosquito detection, air environmental data are taken continuously, and, if they are suitable for mosquito development, pictures of the studied area are taken. Later on, the CNN, trained for mosquito detection, classifies such pictures, obtaining for each picture, the probability of containing a mosquitoes and the probability of belonging to the mosquito species the CNN was trained with. As before, by combining the environmental information with the pictures classification, the probability of having detected mosquitoes as well as the probability, for such mosquitoes, of belonging to the studied mosquito species, are computed.

Notice that the fact of combining environmental data together with pictures classified by an algorithm based on AI, increases significantly the probabilities of successfully detecting mosquitoes and puddles with larvae.

Keeping in mind its role, the designed system is composed of two different structures: the first one is the air structure which can be placed in a drone, rover or statically in the top of a construction, while the second one is the ground structure which must be situated in water.

Regarding the air system structure, as seen in Figure 5.1, it consists of a *LattePanda* Windows 10 development board in which four elements are integrated: a web camera to take low cost pictures, a sensor of air temperature and air humidity to get environmental data, a GPS to know the geographic coordinates of the pictures taken and a Bluetooth adapter to collect water environmental data.

Concerning the ground system structure, as seen in Figure 5.1, it is made of an *Arduino* UNO board in which three elements are integrated: a water depth sensor to determine the depth of puddles, a water temperature sensor to know the puddle temperature and a Bluetooth module to transmit water environmental data.

For simplicity, the communication between the air system and ground system has been

effectuated through the use of Bluetooth, a wireless technology. However, this communication could be carried out using either ZigBee or LoRa technology. In fact, the ground system is always transmitting while the air system captures the transmitted data by the former when passing near.

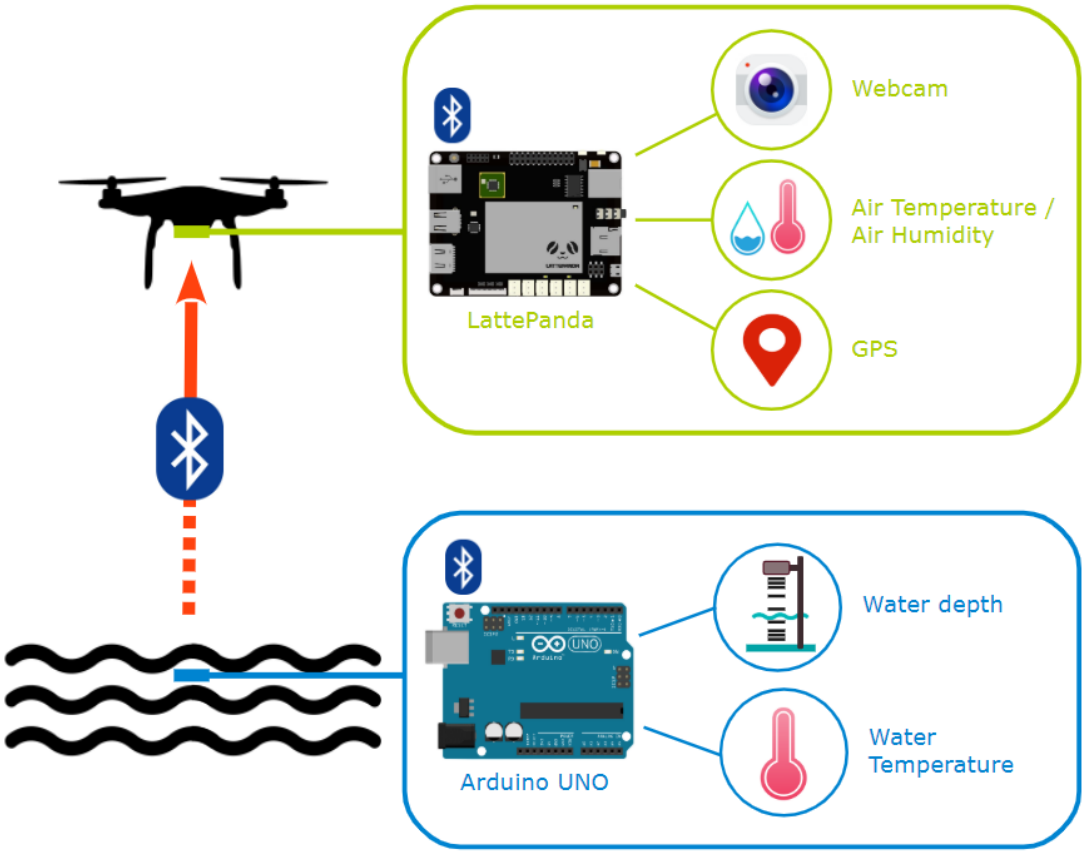


Figure 5.1: Schema of the designed system

5.3. Environmental conditions

To implement the system previously defined, first, the conditions for larval growth and mosquito development must be determined for puddle and mosquito detection respectively. Knowing these conditions is equivalent to know some characteristics of the water where larvae grow and to know the temperature and humidity of the air in which mosquitoes fly.

5.3.1. Larval growth condition

The conditions for larval growth that are taken into account for the next Sections, are summarized in Table 5.1. Notice that some habitat characteristics have a value in parenthesis corresponding to the modus or the central tendency obtained from most frequent data. Moreover, some of these characteristics has a qualitative scale between 1, meaning low and 5, meaning high.

	Aedes albopictus [63] [64] [65] [66] [67]	**Anopheles atroparvus** [62]	**Culex pipiens** [62]
Oviposition site	water surface	water surface	water surface
Hibernation state	egg	female	female
Number of generations within year	multivoltine	multivoltine	multivoltine
Water depth [cm]	5-(10)-25	3-(10)-120	10-(10)-40
Water transparency (scale)	1-(2)-5	1-(3)-5	2-(3)-3
Water pH	5.2-8.35	6.2-(8.3)-14.1	8.2-(9.0)-9.0
Water temperature [℃]	20-(25)-30	6.2-(24.1)-33.2	11.8-26.3
Water surface cover (scale)	1-(3)-5	1-(1)-5	1-(1)-3
Shading (scale)	1-(4)-5	1-(3)-5	1-(2)-2

Table 5.1: Larval growth conditions for the studied mosquito species

5.3.2. Mosquitoes development conditions

The conditions for mosquito development that are taken into account for the next Sections, are summarized in Table 5.2.

	Aedes albopictus [68] [69]	**Anopheles atroparvus** [70] [71]	**Culex pipiens** [72] [71]
Air temperature [℃]	15-35	10-42	15-32
Air humidity [%]	30-90	40-90	40-90

Table 5.2: Mosquitoes development conditions for the studied mosquito species

5.4. Ground system elements

As already explained in Section 5.2., the ground system structure implies the use of a water temperature sensor, a water depth sensor and a Bluetooth adapter. In this section, their functioning is going to be described, and finally, the way in which these elements have been integrated into an *Arduino* UNO board is going to be explained.

5.4.1. Water temperature sensor

The water temperature sensor used in this case is the one corresponding to the model *RS PRO* 3 wire PT100 (see Ref. [73]). This platinum sensor behaves as a variable resistor that changes its value depending on the water temperature according to Equation 5.1 (where R represents the variable resistor and T, the temperature). Indeed, this sensor attains it nominal resistor value at 0 ℃ and it has a platinum temperature coefficient defined by α whose value is 0.00385 $\Omega/\Omega/$℃.

$$R = 100(1+\alpha T) = 100(1+0.00385T) \quad \rightarrow \quad T = \frac{\frac{R}{100}-1}{\alpha} = \frac{\frac{R}{100}-1}{0.00385} \tag{5.1}$$

Equation 5.1 shows that it is needed to know the PT100 sensor resistor value to compute the water temperature. To determine the sensor resistor value, a simple circuit as the one of Figure 5.2 has been built where V is the DC voltage supply of 5 V, $R1$ is equal to 100 Ω and $R2$ corresponds to the sensor resistor. Therefore, through a voltage divider and reading the voltage value present in $R2$, $R2$ value can be computed applying Equation 5.2 and, in this way, the water temperature value. Notice that in Equation 5.2, V_{ADC} represents the voltage value present in $R2$ and read by an analog pin of *Arduino* UNO board.

$$V_{ADC} = \frac{V \cdot R_2}{R_1 + R_2} \rightarrow R_2 = \frac{V_{ADC} \cdot R_1}{V - V_{ADC}} \tag{5.2}$$

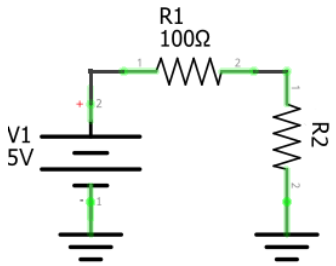
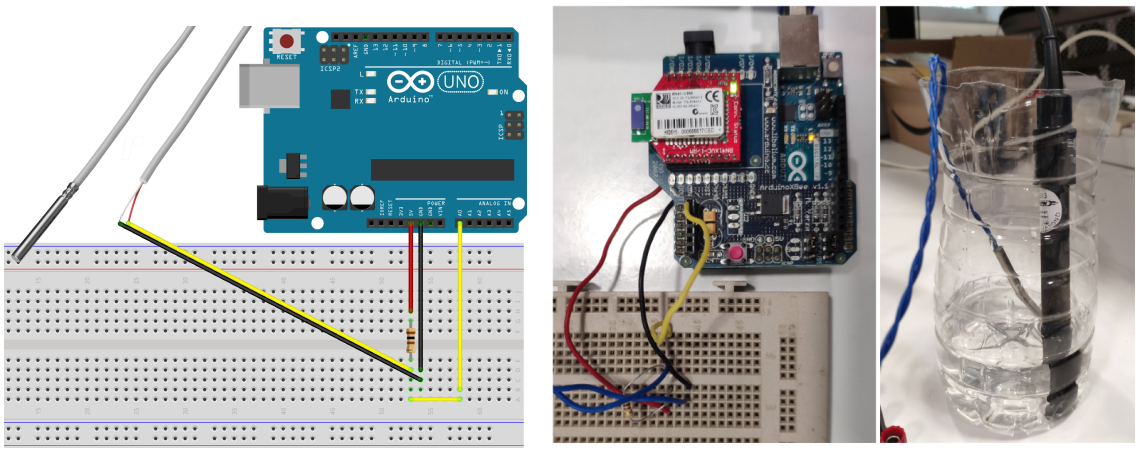


Figure 5.2: Sensor circuit schema

Once known how to determine the water temperature value from a PT100 sensor, in order to get this value automatically, the previous equations have been implemented in the *Arduino* script called "groundMeasurements.ino" (see Appendix K for more details).

Figure 5.3(a) proves that the theoretical circuit of Figure 5.2 can be built with real components. Indeed, it is only about: using *Arduino*'s 5 V as the circuit voltage supply, substituting $R2$ directly by the PT100 sensor and using the A0 analog pin to get the voltage of $R2$. The use of an *Arduino* UNO board and a protoboard is already taken for granted. In parallel to Figure 5.3(a), Figure 5.3(b) shows the final resulting setup with real components.



(a) Schema view

(b) Real view

Figure 5.3: *Arduino* UNO and PT100

5.4.2. Water depth sensor

A water depth sensor, in charge of monitoring the water level in a container or puddle, can be built in different ways. For simplicity, in this project, this sensor has been built using an ultrasonic sensor together with a stick with a given height.

The ultrasonic sensor used for building the depth sensor is the one corresponding to the model HC SR-04 (see Ref. [75]). In fact, it consists of two parts: a transmitter and receiver which, through the use of transducers, convert ultrasound waves to electrical signals and vice versa. The transceiver vibrates and creates an ultrasonic wave that is transmitted and travels until it hits an object and it is reflected back to the receiver (see Fig. 5.4 illustrating this principle). The time interval between the signal being sent and received depends on the distance the ultrasonic wave travels until it is reflected.

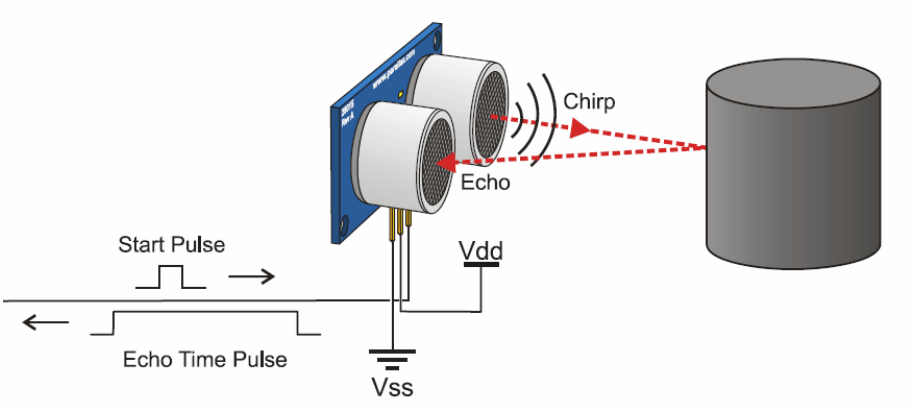


Figure 5.4: Ultrasonic working principle [74]

Therefore, by applying Equation 5.3, where v is the sound speed (340 m/s), d the distance in only one direction and t the time spent by the signal to reach the water surface and be back from it, the distance between the ultrasonic sensor and the water surface can be determined. Notice that t is divided by two in order to determine the time spent by the wave to reach the water surface, that is the time spent traveling only one direction. If Equation 5.3 is reorganized to get d in centimeters, Equation 5.4 is obtained.

$$v = \frac{2d}{t} \quad \rightarrow \quad d = \frac{v \cdot t}{2} \quad \rightarrow \quad d = \frac{340t}{2} \tag{5.3}$$

$$d[cm] = \frac{340[\frac{m}{s}] \cdot t[\mu s] \cdot \frac{1}{10^6}[\frac{s}{\mu s}] \cdot \frac{100}{1}[\frac{cm}{m}]}{2} \quad \rightarrow \quad d[cm] = \frac{340t}{2 \cdot 10^4} \tag{5.4}$$

Nevertheless, with the previous equations, the distance known is the one from the ultrasonic sensor to the water surface and, what it is desired, is to estimate the water depth. To get it, it is only necessary to know the height of the stick where the ultrasonic sensor is placed (in this case, the used stick has a height of 38.5 cm) and to apply Equation 5.5 where d is the distance above water and d' the one below water (see Fig. 5.5).

$$d' = h - d \rightarrow d'[cm] = 38,5 - d[cm] \quad (5.5)$$

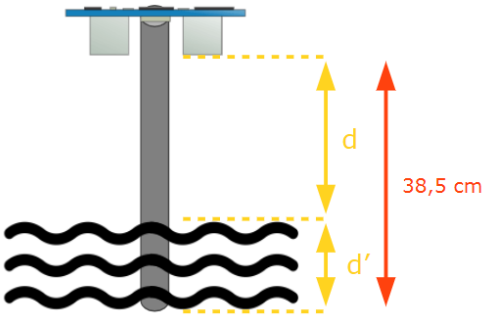
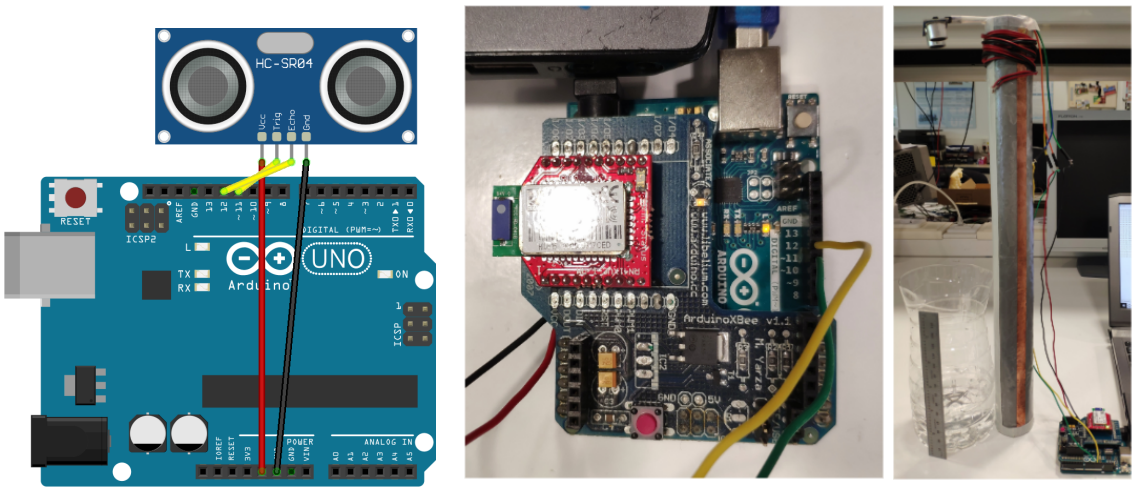


Figure 5.5: d and d' definition

Once known how to determine the water depth from a HC-SR04 sensor, in order to get this value automatically, the previous equations have been implemented in the *Arduino* script called "groundMeasurements.ino" (see Appendix K for more details).

Figure 5.6(a) shows how to connect the sensor pins to *Arduino* pins. Indeed, the sensor receives 5 V Arduino voltage supply through *Vcc* pin and it is connected to ground through *Gnd* pin. Furthermore, through digital pin 11, *Arduino* board send an electric signal (*Vss* in Fig. 5.4) to the sensor *Trig* pin and immediately after, the sensor converts the signal to ultrasound waves. In a similar way, the sensor converts the reflected ultrasound waves to an electrical signal (*Vdd* in Fig. 5.4) that sent from its *Echo* pin to the digital pin 12 of *Arduino* board. Hence, by knowing the elapsed time between the generated electrical signal and the received one, the water depth is estimated. In parallel to Figure 5.6(a), Figure 5.6(b) shows the final resulting setup with real components.



(a) Schematic view

(b) Real view

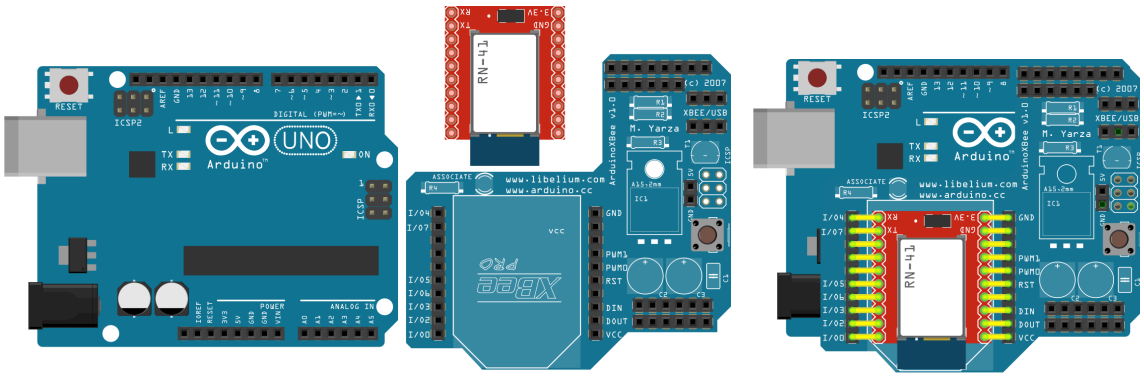
Figure 5.6: *Arduino* UNO and HC SR-04

5.4.3. Bluetooth

Implementing a Bluetooth communication with the RN-41 module (see Ref. [76]) is almost immediate. In fact, this module is fitted in an *Arduino* XBee module, already prepared to be used with *Arduino* UNO board.

When RN-41 is fitted in *Arduino* XBee module and at the same time, this former is fitted in *Arduino* UNO board (see Fig. 5.7), RN-41 turns on a yellow led which stays blinking until a connection is established. To establish a connection with the Bluetooth module from a PC, by default, it is only required to activate the Bluetooth of such PC, to search for a device called ”RNBT-7CED” and to pair both devices. Then, when the PC is close enough to RN-41 and the PC COM port for receiving data through Bluetooth, is open, automatically, the connection is established and the corresponding data are transferred. For more information about the setup of this process, refer to Appendix C.2..

It should be noted that any *Arduino* code have to be uploaded to the *Arduino* board before connecting the module to such board. After uploading the code, the board should be disconnected from any power supply and, at this moment, the Bluetooth module can be fitted in the *Arduino* pins and then, the *Arduino* board can be powered again.



(a) Separate view (b) Integration view

Figure 5.7: *Arduino* UNO and *Arduino* XBee (with RN-41)

5.4.4. Integration

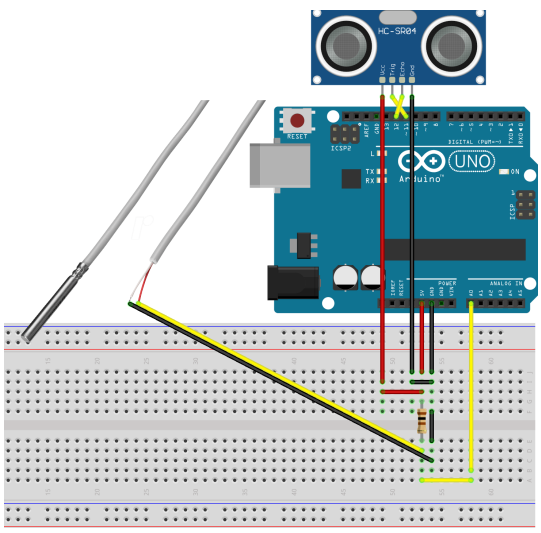
If Section 5.4.1., 5.4.2. and 5.4.3. have been well understood, the integration of these elements together into an Arduino UNO board is almost immediate.

Thus, the water temperature sensor and the water depth sensor have to be connected in the same *Arduino* UNO board in the way it was explained in Section 5.4.1. and 5.4.2., respectively. However, the only connection that changes, is the connection that these sensors have with 5 V pin of *Arduino* UNO board, as there is only one and both sensors need to be connected to it. As Figure 5.8(a) shows, this issue is solved by connecting both sensors in parallel. The same has been done with the ground pin to maintain a certain consistency, although *Arduino* UNO board has two ground pins.

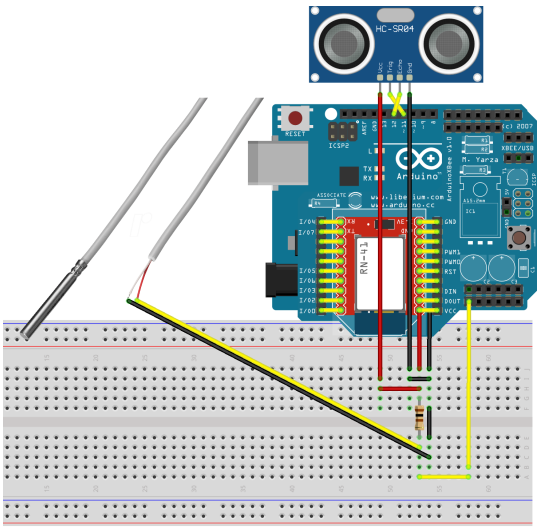
Before fitting *Arduino* Xbee module and RN-41 into *Arduino* UNO board, the *Arduino* script called ”groundMeasurements.ino” has been uploaded to *Arduino* UNO board (see Appendix K for more details).

Now, *Arduino* Xbee module and RN-41 can be fitted into *Arduino* board. Nevertheless, the wire connected previously to A0 analog pin, has to be switched to *Arduino* Xbee A0 analog pin (which is exactly at the same place), as Figure 5.8(b) displays. This is due to the fact that some pins of *Arduino* UNO board are occupied by the pins of *Arduino* Xbee module.

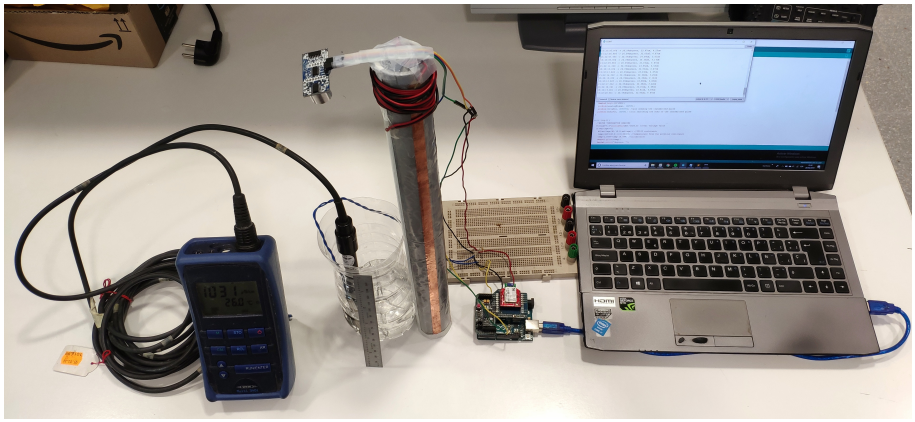
Finally, the final setup with real components is shown in Figure 5.8(c).



(a) Schematic view - First phase



(b) Schematic view - Second phase



(c) Real view

Figure 5.8: Integration of previous elements with *Arduino* UNO

5.5. Air system elements

As already explained in Section 5.2., the air system structure implies the use of an air temperature and air humidity sensor, a web camera, a GPS and a Bluetooth adapter. In this section, only the function of the sensor is going to be detailed as the other elements works directly through USB port, making the integration process much easier. Finally, the way in which these elements have been integrated into a *LattePanda* board through *Python* scripts, is going to be explained.

5.5.1. Air temperature and air humidity sensor

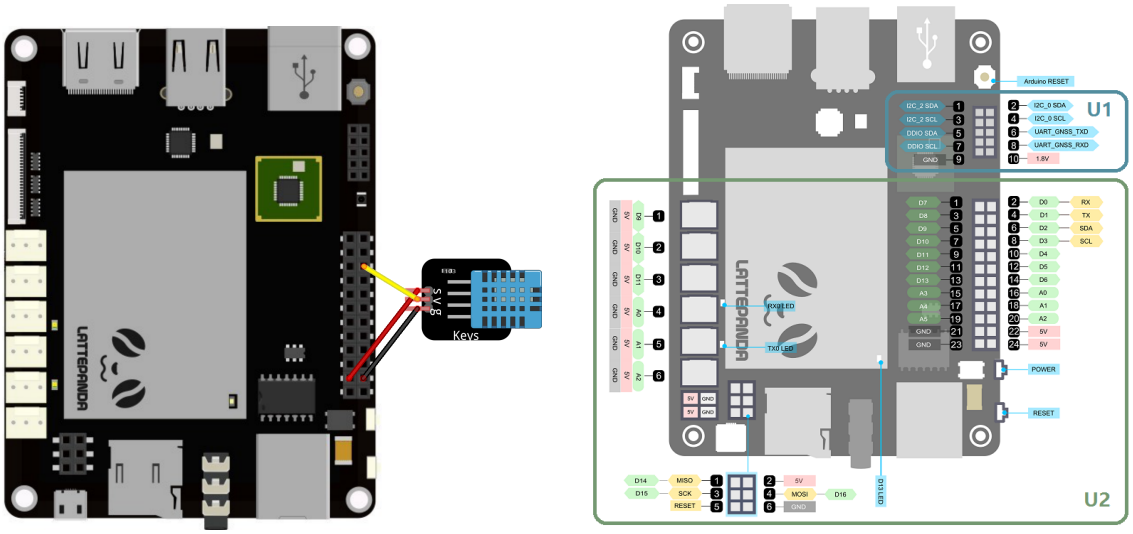
The temperature and humidity sensor used in this case is the DHT11 module (see Ref. [77]). In fact, the DHT11 comes with a dedicated NTC to measure temperature and an 8-bit micro-controller to output the values of temperature and humidity as serial data. Moreover, it is also factory calibrated and hence easy to interface with other micro-controllers.

Regarding its performance, it can measure temperature from 0 ℃ to 50 ℃ and humidity from 20% to 90% with an accuracy of ±1 ℃ and ±1%, respectively, which is enough to know if environmental conditions are suitable for mosquito development.

To integrate this sensor with *LattePanda* board, as Figure 5.9(a) and 5.9(b) show, DHT11 pin *S* (voltage) should be connected to *LattePanda* pin 21, DHT11 pin *G* (ground) to *LattePanda* pin 22 and DHT11 pin *V* (data) to *LattePanda* pin 4.

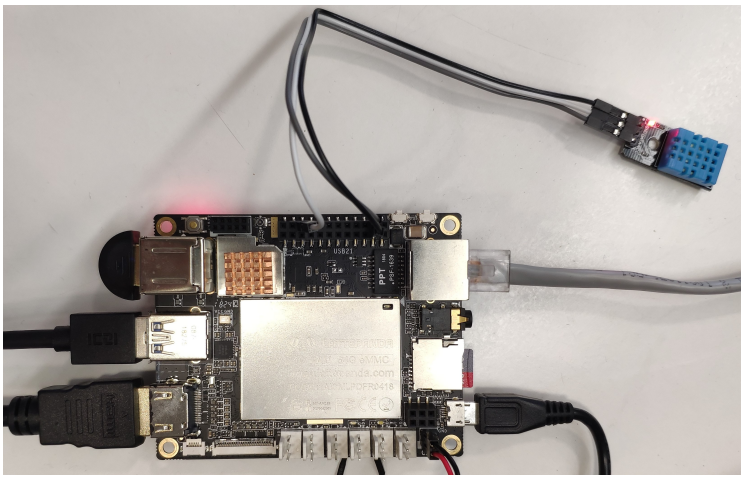
Then, taking advantage that *LattePanda* board has a built-in *Arduino* Leonardo, the *Arduino* script called "airMeasurements.ino" has been uploaded to such board (see Appendix J for more details of the code). It should be noted that this script uses a library called "DHT.h" to be able to communicate with such sensor. To know how to install this library, refer to Appendix C.3..

Finally, the real integration of this sensor with a *LattePanda* board can be observed in Figure 5.9(c).



(a) Pins connection view

(b) Pins schema view [78]



(c) Real view

Figure 5.9: Connection between DHT11 and *LattePanda* board

5.5.2. *Python* scripts for gathering air data

Although a script is not a tangible element as the sensor before, the *Python* scripts called "puddle_process.py" and "mosquitoes_process.py" (see Appendix E and F for more details of the codes) play a fundamental role in the air system for detecting puddles and classifying mosquitoes, respectively. To understand it, let's define the function, the inputs and outputs of both scripts.

5.5.2.1. Function

The basic objective of both scripts is to take pictures with the camera and process them through the use of AI (as explained in Chapter 4), in combination with the positioning given by the GPS and the temperature and humidity recorded by the temperature and humidity sensor. Thus, through this combination of image processing and environmental data, the probability of detecting puddles and mosquitoes from pictures increases significantly.

For the case of the puddle detection script, for each picture taken, the temperature and the humidity is recorded and the GPS positioning is associated with such picture. Immediately after, the picture is assessed by the AI algorithm to know the probability of having a puddle in it. Finally, all these numerical data are graphically depicted in a Google Earth map using the GPS positioning data.

Notice that, in the case of puddle detection, the temperature and humidity of the air, although they are recorded, they are not used later on, as what it is desired is to estimate the probability of finding larvae within the puddle, that is, within the water. In other words, for larval detection, air conditions do not give enough information, that is why puddle data are later on combine with ground data.

For the case of mosquito detection script, as before, for each picture taken, the temperature and the humidity is recorded and the GPS positioning is associated with such picture. Once all the desired pictures are taken, thereafter, these are assessed by the AI algorithm to know the probability of containing mosquitoes in it and as well to determine the probability for such mosquitoes of belonging to the studied species. After sweeping a given area, the analysis of such probabilities leads to an estimated mosquito density. Such image processing is combined with the temperature and humidity recorded data for a better mosquito detection. Finally, all this numerical data are graphically depicted in a Google Earth map using the GPS positioning data.

In this last case, the reason for processing pictures after being taken is that each picture taken has to be split into smaller pictures and actually, these small pictures are the ones processed by the AI algorithm. This increases considerably the number of pictures to process, making the processing of the originally taken picture specially slow. As it is obvious, it is not possible to sweep an area within a relatively short time and to process the taken pictures at the same time. Nevertheless, it should be noted that mosquito pictures are split equally in smaller pictures in such a way their resolution corresponds to the size of a mosquito (this resolution value is later on computed). Thus, after processing these cropped pictures, the mosquito density per picture taken can be estimated. In fact, Figure 5.10 illustrates the idea of cropping mosquito images.

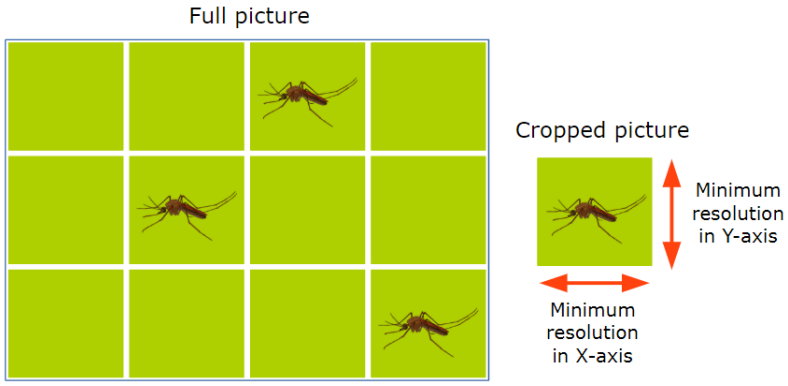


Figure 5.10: Schema of how mosquito pictures are cropped

5.5.2.2. Inputs

Both scripts have 4 inputs in common, which are:

- the folder into which output files are going to be saved
- GPS COM Port
- Camera COM Port
- the number of minutes it is desired to run the script

Moreover, the mosquito detection script has an added input, which is the width in pixels of the cropped pictures.

5.5.2.3. Outputs

Both scripts have 5 output files in common, which are:

- ”airMeasurements.csv”: comma-separated values (CSV) file in which it is saved, for each picture, its name, the date and time in which it was taken, its GPS latitude, its GPS longitude, the current temperature, the current humidity and the current temperature index.
- ”classification.csv”: CSV file in which it is saved all the probabilities determined by the AI algorithm for each taken picture.
- ”gpsData.txt”: text file in which it is saved the latitude and longitude directly given by the GPS (it is only used to track the GPS).
- ”puddle_location.kml” or ”mosquitoes_location.kml”: *Google Earth* file in which the location of puddles and mosquitoes is graphically saved.
- ”results.csv”: CSV file in which it is saved the final label assigned to each picture after processing and analyzing it.

Moreover, the mosquito detection script has an added output called ”pictures_classification.csv” which is a CSV file that gathers the number of mosquitoes found per picture, classified by specie.

5.5.3. *Python* script for gathering ground data

The *Python* script called ”save_bluetooth_data.py” (see Appendix H for more details of the code) plays a essential role in the puddle detection system. To understand it, let’s define the function, the inputs and outputs of the script.

5.5.3.1. Function

The main objective of this script is to capture the last 10 data samples measured by the ground system when a Bluetooth connection with RN-41 module is established.

This script was designed thinking that *LattePanda* board is fitted in a non-statical structure (i.e. a drone or a rover) and that there is a single ground systems monitoring a given puddle. In this way, RN-41 module is not always paired with the *LattePanda* Bluetooth, as *LattePanda* board is always moving. That is why the script waits indefinitely until it founds the RN-41 module and establishes a connection. After gathering the corresponding 10 samples and thus, after capturing the needed data, the script is stopped.

5.5.3.2. Inputs and outputs

This script has a single input which is the Bluetooth COM port through which data are received, and a single output which is a text file called ”groundMeasurements.txt. In this file, the data related to the water temperature, the depth of the puddle into which the ground sensor was placed and the distance between the ultrasonic sensor to puddle water, are saved.

5.5.4. *Python* script comparing ground and air data

The *Python* script called ”puddle_and_btb_data_comparison.py” (see Appendix I for more details of the code) plays a basic role, combining air and ground data, to assess the suitability of the environment for larval growth. To understand it, let’s define the function, the inputs and outputs of the script.

5.5.4.1. Function

The main objective is to merge the ground and air data and to modify, if necessary, the CSV file ”results.csv” accordingly.

The script takes, from the CSV file ”airMeasurement.csv”, the name of each picture and the date and time each one was taken and, from the CSV file ”results.csv”, the label assigned to each picture. In case the label assigned is ”puddle”, the picture is assessed again but this time with the ground data. This means, the average temperature and the average humidity are computed within an interval of $\pm$ 10 minutes from the moment the picture was taken, using data from ”groundMeasurements.txt”. Finally, using the larval growth conditions exposed in Table 5.1, it is determined if the label was correctly been assigned or it has to be changed to ”misc”.

5.5.4.2. *Inputs and outputs*

This script has two inputs which are the folder into which output files with the air data have been saved and the folder into which output file with the ground data have been saved.

Regarding the outputs, taking into account to the ground data, if necessary, the script modifies the CSV file ”results.csv” by changing the final label assigned to the pictures that require it.

5.5.5. Integration

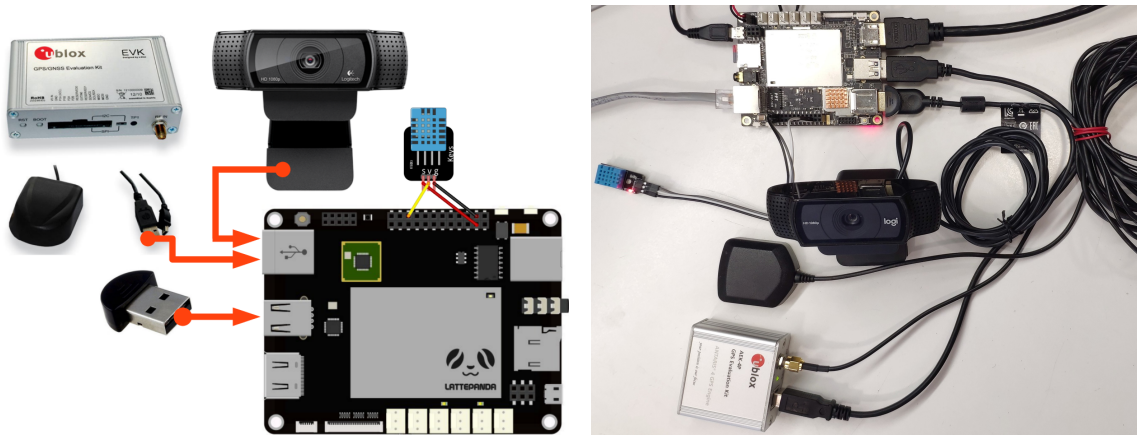
If Section 5.5.1., 5.5.2., 5.5.3. and 5.5.4. have been well understood, the integration of these elements together into *LattePanda* board is almost immediate.

In fact, first, the previous scripts have to be saved into such board and the needed *Python* packages to execute these scripts, have to be installed (see Appendix C.1. to proceed with the packages installation). Next, the camera USB, the GPS USB and the Bluetooth USB have to be connected to *LattePanda* board and their COM port has to be determined in order to correctly define the inputs of such scripts, as well as the folder into which their output files are going to be saved. Finally, three batch files have to be generated in order to execute automatically, when the board turns on, the scripts related to puddle detection, mosquito detection and ground data gathering. To know the content of such batch files and the folder into which they should be saved, refer to Appendix C.4..

Concerning the temperature and humidty sensor, it has to be connected the same *LattePanda* board in the way it was explained in Section 5.5.1. without forgetting to upload to the board the *Arduino* script associated to such sensor.

Notice that it is taken for granted that the software executing these scripts is already installed, if it is not the case, refer to Appendix C.1. so to prepare *LattePanda* board for the designed system. Furthermore, the script merging the ground and air data (see Appendix I) is not executed automatically and it has to be executed after gathering the required data.

After following the previous steps, the final setup with real components is such as the one shown in Figure 5.11.



(a) Schematic view (b) Real view

Figure 5.11: Integration of the previous elements with *LattePanda*

5.6. Camera choice

A camera is needed to take real stagnant water pictures and real mosquito pictures, which are of interest to be classified by the two trained CNNs of Section 4.

The choice has been done according to two requirements: first, the camera must be able to see mosquitoes, which implies determining the resolution of a mosquito in a picture, and second, a web-camera must be chosen, as it is the cheapest option in the market and what is desired is to design the cheapest possible system.

5.6.1. Resolution of a mosquito in a picture

To determine the resolution of a mosquito in pixels per centimeter in a picture, first the minimum number of pixels that a picture of a mosquito has to contain, must be fixed. This has to be done in such a way that the CNN is still able to distinguish correctly between the species of mosquitoes it was trained with.

For this purpose, the pictures of the mosquito data set, used in Section 4.3., have been resized to different dimensions, from the original ones that are 224 x 224 pixels to 32 x32 pixels, going through 128 x 128, 80 x 80, 64 x 64, 56 x 56, 52 x 52 and 48 x 48 and have been saved into different data sets according to the pictures dimensions.

Thus, *Inception* model has been retrained with these different data sets separately for which different final test accuracies have been obtained, which has lead to the elaboration of Figure 5.12. In fact, as it shows, the final test accuracy decreases exponentially with the number of pixels per picture width.

From Figure 5.12, it is established around 70% the minimum expected final test accuracy for two reasons. The first reason is that a final test accuracy of 70% is pretty less than the original one around 79% obtained with 224 pixels and, at the same time, it is pretty much than a final test accuracy of 50%, which is equivalent to fail classifying 1 over 2 pictures. Concerning the second reason, lots of pictures are going to be taken and classified, which will compensate any misclassification for certain pictures.

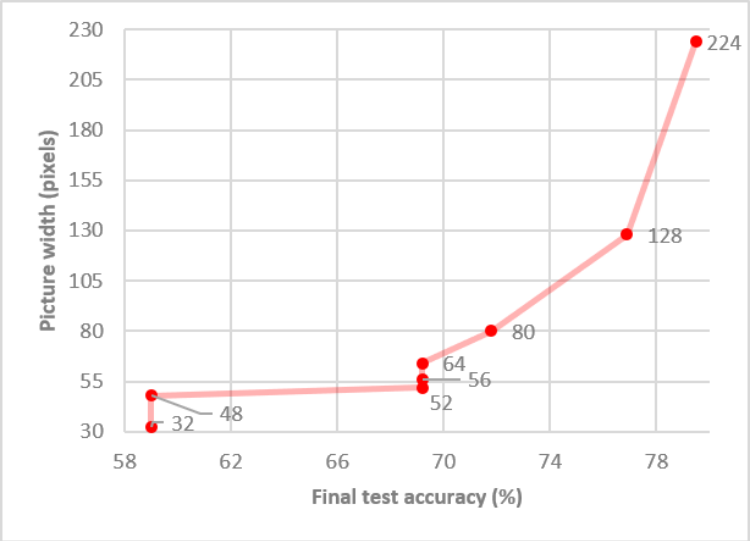


Figure 5.12: Final test accuracy vs Pixels in picture width

Therefore, a final test accuracy of 70% has been considered enough to classify mosquitoes correctly and detecting when there is no mosquito in a picture. Indeed, this accuracy corresponds to either 64, 56 or 52 pixels per picture width. To be prudent, the minimum number of pixels that a picture of a mosquito must contain, has been settled taking into account the highest number of pixels fulfilling the established minimum final test accuracy, which can be summarized in considering 64 pixels.

Once the minimum number of pixels occupied by a mosquito in a picture is established, the real size of the studied mosquitoes must be determined and this is what is exposed in Table 5.3. What this Table shows is that the minimum average size of these mosquitoes is around 5 mm.

Moquito specie	aedes albopictus	anopheles atroparvus	culex pipiens
Size	5 - 10 mm [48]	5 -10 mm [50] [51]	4 - 10 mm [52]

Table 5.3: Size of the studied mosquito species

Finally, by considering an average mosquito size of 0.5 cm and the dimensions of mosquito picture such as 64 x 64 pixels, the resolution of a mosquito in a picture can be determined as shown in Equation 5.6.

$$res_{x_{mosquito}} = res_{y_{mosquito}} = res_{mosquito} = \frac{size_{mosquito}}{dim_x} = \frac{64}{0.5} = 128[pixels/cm] \quad (5.6)$$

5.6.2. Assessed cameras

Once the resolution of a mosquito in a picture is known, a web-camera must be chosen to take real pictures of puddles and mosquitoes as well.

To achieve it, the best web-cameras currently existing in the marketplace have been listed and evaluated according to the pixels number in X axis ($x_{webcam}[pixels]$), pixels number in Y axis ($y_{webcam}[pixels]$), horizontal angle ($\phi[^\circ]$), vertical angle ($\theta[^\circ]$), horizontal distance from the camera ($d_{webcam_{min}}[cm]$) and price (see Table 5.4 and 5.5).

Model	x_{webcam} $[pixels]$	y_{webcam} $[pixels]$	$\phi[^\circ]$	$\theta[^\circ]$	**Price**
Aukey PC-LM1-EU [53]	1920	1080	55	55	30,99 €
Besteker 920C-001 [54]	2048	1536	80	64	59,99 $
Intel Real Sense D435 [55]	1920	1080	69,4	42,5	179 $
Logitech Brio [56]	4096	2160	78	65	181,98 €
Logitech C615 HD [57]	1920	1080	70,42	43,3	89,90 €
Logitech C920 HD Pro [58]	2304	1296	70,42	43,3	67,99 €
Logitech C922 Pro Stream [59]	1920	1080	70,42	43,3	84,98 €
Microsoft LifeCam HD-3000 [60]	1280	720	61	34,3	27,37 €
Razer Kiyo [61]	2688	1520	72,4	44,7	89,99 $

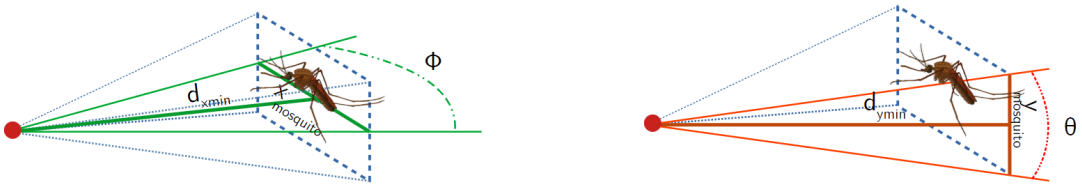
Table 5.4: Features of the selected web-cams

Using the previous mosquito resolution and the web-camera resolution in pixels (x_{webcam}

and y_{webcam}), the size of a mosquito such as it would appear in the selected cameras pictures (i.e. composed of $x_{mosquito}[cm]$ and $y_{mosquito}[cm]$), has been computed according to Equation 5.7.

$$x_{mosquito} = \frac{x_{webcam}}{res_{mosquito}} \qquad y_{mosquito} = \frac{y_{webcam}}{res_{mosquito}} \tag{5.7}$$

Then, after knowing the size that a mosquito occupies in the pictures of each considered web-camera, the minimum distance between web-camera and mosquito in x-axis and y-axis must be calculated. To accomplish it, an approximation must be done: the web-camera must be considered as a point (a red point in Fig. 5.13). Subsequently, the minimum distance from the web-camera to the mosquito x-axis, $d_{x_{min}}$, can be known by applying Equation 5.8 which uses the web-camera horizontal angle, ϕ and the variable $x_{mosquito}$. In the same way, the minimum distance from the web-camera to the mosquito y-axis, $d_{y_{min}}$, can be known by applying Equation 5.9 which uses the web-camera vertical angle, θ and the variable $y_{mosquito}$.



(a) Horizontal field of view

(b) Vertical field of view

Figure 5.13: Web-camera fields of view

$$\tan\left(\frac{\phi}{2}\right) = \frac{x_{mosquito}}{d_{x_{min}}} \quad \rightarrow \quad d_{x_{min}} = \frac{x_{mosquito}}{\tan\left(\frac{\phi}{2}\right)} \tag{5.8}$$

$$\tan\left(\frac{\theta}{2}\right) = \frac{y_{mosquito}}{d_{y_{min}}} \quad \rightarrow \quad d_{y_{min}} = \frac{y_{mosquito}}{\tan\left(\frac{\theta}{2}\right)} \tag{5.9}$$

The results of the previous equations are presented in Table 5.5. Using this Table, now it is only about comparing the minimum distance given by web-cams specifications with the minimum of the minimums distances computed (i.e. minimum of $d_{x_{min}}$ and $d_{y_{min}}$).

For the cases where the specified minimum distance from which the camera is able to capture, is higher than the one computed, the web-cams are automatically discarded, that is the case of *Aukley* PC-LM1-EU, *Intel* Real Sense D435, *Logitech* C922 HD Pro and *Microsoft* LifeCam HD-3000. However, some web-cameras do not have a minimum specified distance, that is why their minimum distance is not available in Table 5.5.

The only web-camera having a specified minimum distance lower than the one computed, is *Logitech* C920 HD Pro. Moreover, it is cheaper than *Logitech* Brio, *Logitech* C615 HD and *Razer* Kiyo, which is the reason why these web-cams are also discarded.

Hence, there are two models remaining: *Besteker* 920C-001 and *Logitech* C920 HD Pro. Although the first one is cheaper, its minimum distance is unknown and the resolution of the second case is better, leading to finally choose *Logitech* C920 HD Pro.

Model	$d_{webcam_{min}}$ $[cm]$	$x_{mosquito}$ $[cm]$	$y_{mosquito}$ $[cm]$	$d_{x_{min}}$ $[cm]$	$d_{y_{min}}$ $[cm]$
Aukey PC-LM1-EU	30	15	8,44	14,41	8,10
Besteker 920C-001	?	16	12,00	9,53	9,60
Intel Real Sense D435	11	15	8,44	10,83	10,85
Logitech Brio	?	32	16,88	19,76	13,24
Logitech C615 HD	?	15	8,44	10,63	10,63
Logitech C920 HD Pro	2,54	18	10,13	12,75	12,75
Logitech C922 Pro Stream	10	15	8,44	10,63	10,63
Microsoft LifeCam HD-3000	30	10	5,63	8,49	9,11
Razer Kiyo	?	21	11,88	14,35	14,44

Table 5.5: Determined parameters from the features of the selected web-cams

CHAPTER 6. VERIFICATION AND VALIDATION OF THE MOSQUITO PREVENTION SYSTEM

Keeping in mind the designed system, in this chapter, the verification and validation of the system architecture is going to be performed. In other words, the mosquito density estimation as well as the air and ground data fusion for larval detection are going to be verified. Then, the suitability of the chosen camera for puddle and mosquito detection is also going to checked. Next, the ground sensors accuracy is going to be assessed. Finally, the validation of the system for detecting mosquito larvae, is going to be carried out.

6.1. Aim

The main purpose of this chapter is to demonstrate, through the system verification and validation, how feasible the designed system architecture is and how easy it is to gather all the needed information from different devices to fight against mosquitoes and the diseases they transmit. Moreover, it is intended to prove that AI together with the quantification of environmental data can increases significantly the probability estimated by the system and thus, its reliability.

Above all, what it is desired is to highlight the potential of this mosquito prevention system either for reducing the number of human deaths due to malaria in Africa or for monitoring disturbing plagues of mosquitoes. In the end, this system could ease the cohabitation between humans and insects.

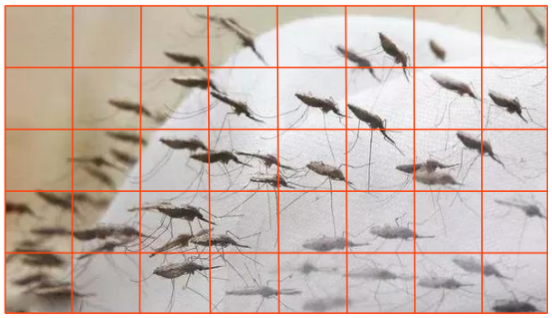
6.2. System verification

To check the system works as expected, three different verifications are going to be carried out. First, through pictures taken from the Internet, it is going to be verified that the designed system is able to estimate the mosquito density present in such pictures. Then, through the air and ground data merging, it is going to be proved that, the probability of successfully detecting puddles containing mosquito larvae, improves. Finally, through pictures taken with the chosen camera in a controlled environment, it is going to be checked that the AI algorithm is able to process them as it does for the ones downloaded from the Internet.

6.2.1. Mosquito density estimation verification

To verify that the designed system is able to determine the mosquito density from a picture, two pictures of a bunch of mosquitoes have been downloaded from the Internet and analyzed by the CNN trained for mosquito recognition. These pictures are shown in Figure 6.1(a) and 6.2(a). In fact, the first one contains approximately 43 mosquitoes and the second one 23. Moreover, it has been supposed that the first picture is located E1.987256 °and N41.275336 °, and the other one E1.987257 °and N41.275337 °. By knowing the location of these pictures, the mosquito density can be depicted later on.

(a) First bunch of mosquitoes



(b) Resulting cropping

Figure 6.1: First picture downloaded from the Internet

(a) Second bunch of mosquitoes

(b) Resulting cropping

Figure 6.2: Second picture downloaded from the Internet

Once these pictures downloaded, they have been processed by the *Python* script called "mosquitoes_process.py" (see Appendix F for more details of the code) but without using either temperature and humidity data or real GPS data.

To able to estimate the mosquito number present per picture, the script has cropped each of these pictures into smaller and equal pictures with a size of 64 x 64 pixels (see Fig. 6.1(b) and 6.2(b)). This size is the computed resolution of a mosquito such that the AI algorithm is still able to classify mosquitoes correctly, without going too many times wrong. This has resulted in analyzing and classifying 45 cropped pictures for the first picture and 28 cropped pictures for the second one. (Notice that the resulting number of cropped pictures depends on the number of pixels the original picture has.) The pictures have been classified among the 4 different labels which the CNN was trained with, that is "aedes albopictus", "anopheles atroparvus", "culex pipiens" and "miscellaneous". Consequently, for each cropped picture, a certain probability is assigned to each of these labels. If these probabilities are summed for each picture, they are always equal to 1. The criteria followed to assign the final label to each of these cropped pictures, is such that the label with the highest probability is selected.

Therefore, as soon as each cropped picture has an assigned label, for each original picture, it is possible to count the number of cropped pictures that have the same label, leading thus, to the results of Table 6.1.

Picture	# aedes	# anopheles	# culex	# misc	# mosquitoes
1987256E_41275336N	5	9	25	6	39
1987257E_41275337N	0	17	6	5	23

Table 6.1: Mosquitoes density contained in each picture according to the different studied species

If the real number of mosquitoes present in each picture is compared to the number of mosquitoes detected by the algorithm, it can be seen that the difference between both is quite small, checking that the algorithm can count the number of mosquitoes per picture.

After determining the number of mosquitoes per picture according to the specie they belong, if the GPS coordinates of the picture are known, the number of mosquitoes per location can be plotted such as in Figure 6.3, where mosquitoes of *aedes albopictus* specie are represented with a blue pin, the ones of *anopheles atroparvus* specie with a red pin and the ones of *culex pipiens* specie with a yellow pin. Notice that the size of the pin varies in a logarithmic way with the number of mosquitoes represented. This proves that, if a lot of pictures were taken in a given area and the number of mosquitoes per picture was determined, the mosquito density in a specific zone could be known and plotted.

It should be noted that the script called ”draw_sizepin_from_file.py” (see Appendix G for more details of the code), is in charge of generating *Google* Earth files. Moreover, the way in which pictures are cropped, does not take into account if a mosquito has appeared partially in two different cropped pictures and it has been counted twice. So, if the way in which pictures are cropped, is improved, the density estimation could be more accurate.

Figure 6.3: Supposed location of the pictures taken

6.2.2. Air and ground data merging verification for larval detection

To check the system is able to combine ground and air data to improve the probability of successfully detecting puddles containing larvae, it has been supposed that a picture has been captured and that the positioning obtained at the moment in which the picture was captured, is the one of Table 6.2. Furthermore, it has been assumed that, after processing such picture, the CNN, trained for puddle recognition, has assigned the label ”puddle” to it, thus, obtaining Figure 6.4.

It should be noted that this post processing is executed by the *Python* script called ”puddle_and_btb_data_comparison.py” (see Appendix I for more details of the code).

Picture	Datetime	Latitude	Longitude
1987596E_41275577N.jpg	03/07/2019 15:27:08	41275577N	1987596E

Table 6.2: Content of the file ”airMeasurements.csv”

Figure 6.4: Supposed location of the picture taken

At the same time the supposed picture was captured, the ground measurements data were received, corresponding to the puddle appearing on such picture. These data are the one shown in Code 6.1. As it can be seen, the last column corresponding to the puddle depth shows that there are no more than 2 cm of water, which, according to Table 5.3.1., it is not a suitable condition for larval growth.

```
03/07/2019 15:15:06  24.87 degrees, 34.24 cm, 1.26 cm
03/07/2019 15:15:07  24.87 degrees, 33.90 cm, 1.60 cm
03/07/2019 15:15:08  24.87 degrees, 33.90 cm, 1.60 cm
03/07/2019 15:15:09  24.87 degrees, 33.90 cm, 1.60 cm
03/07/2019 15:15:10  24.87 degrees, 34.74 cm, 1.76 cm
03/07/2019 15:15:11  24.87 degrees, 33.95 cm, 1.55 cm
03/07/2019 15:15:12  24.87 degrees, 34.30 cm, 1.20 cm
03/07/2019 15:15:13  23.64 degrees, 34.31 cm, 1.19 cm
03/07/2019 15:15:14  23.64 degrees, 33.88 cm, 1.62 cm
03/07/2019 15:15:15  23.64 degrees, 34.31 cm, 1.19 cm
```

Listing 6.1: ”Raw data measured by the ground system and received through Bluetooth”

Immediately after, the script checks that the ground measurements were collected within an interval of $\pm$10 minutes from the moment the puddle picture was taken. And then, it modifies the label assigned to such picture, to ”misc” as, although it contains a puddle, it does not fulfill the conditions for larval growth. This label shift is depicted in Figure 6.5.

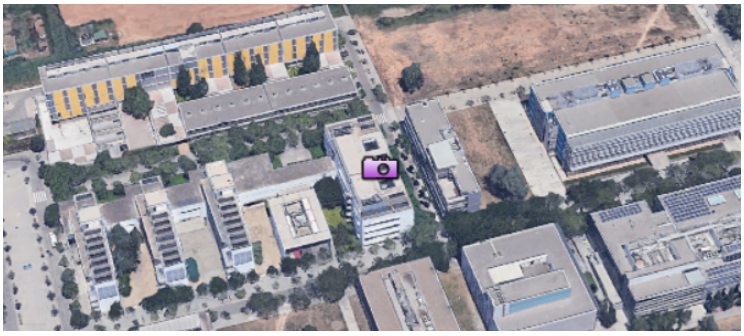


Figure 6.5: Label shift after processing the ground measurements data

6.2.3. Camera verification for puddle detection

To be sure that the CNN, traind for stagnant water detection, is able to identify puddles and stagnant waters in pictures taken by the chosen camera, some pictures of Castelldefels School of Telecommunications and Aerospace Engineering (EETAC) environment containing these targets have been taken with this camera and have been processed by the *Python* script called "theoretical_process.py".

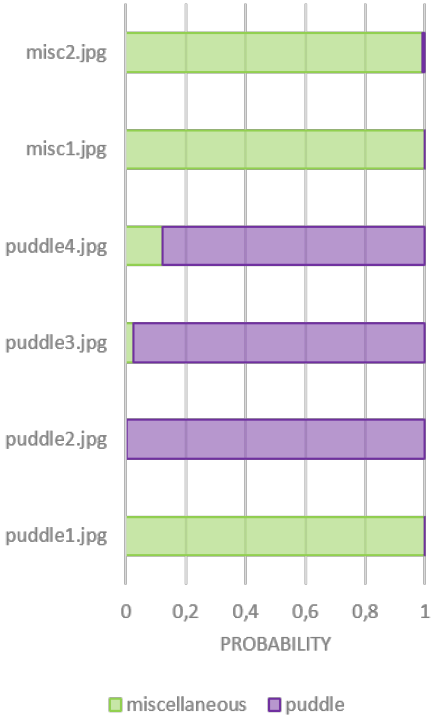
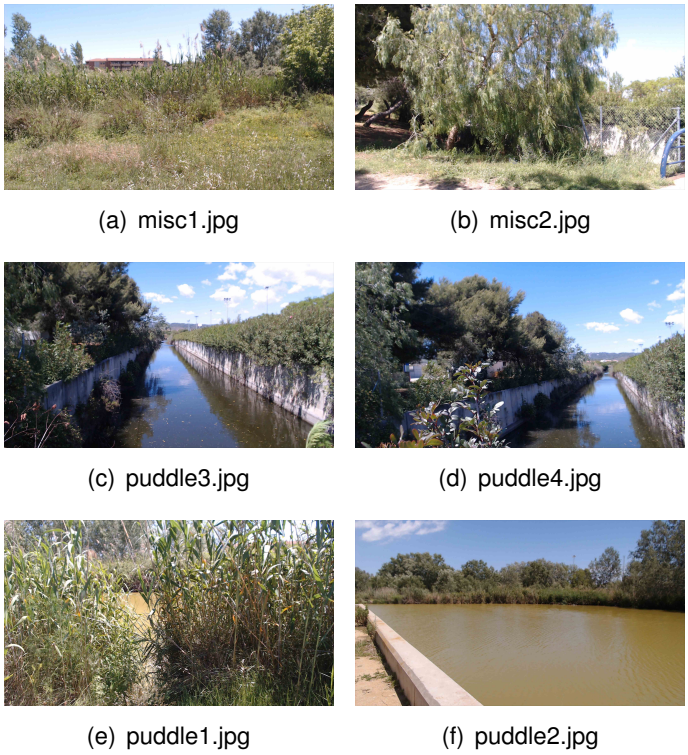


Figure 6.6: Probabilities distribution



(a) misc1.jpg (b) misc2.jpg

(c) puddle3.jpg (d) puddle4.jpg

(e) puddle1.jpg (f) puddle2.jpg

Figure 6.7: Pictures taken with *Logitech* C920 HD Pro

Some of the taken pictures and their corresponding probability are shown in Figure 6.6 and 6.7. There are 6 pictures, 4 of stagnant waters and 2 of the environment of the photographed waters but without any trace of water. Such as Figure 6.6 demonstrates, the detection of puddles is correctly effectuated in 3 out of 4 puddle images, meaning that "puddle" label has a probability around 0.9. The fact that the image "puddle1.jpg", containing stagnant waters, is not well classified ("misc" label probability near 1), is due to the fact that the water is poorly visible and occupies a small image portion, making difficult its detection. However, the two pictures without puddles are well classified in "misc" label (miscellaneous), with a probability near 1.

Although these pictures, representing the different results obtained after processing the taken pictures around the EETAC, are few, they are considered enough to prove the proper functioning of the chosen camera in controlled spaces. Hence, the camera verification for puddle detection has been carried out successfully.

6.2.4. Camera verification for mosquito detection

To be sure that the CNN, trained for mosquito classification, is able to identify mosquitoes in pictures taken by the chosen camera, different pictures have been taken with this camera in the Barcelona neighborhood of Vallcarca and have been processed by the *Python* script called ”theoretical_process.py”.

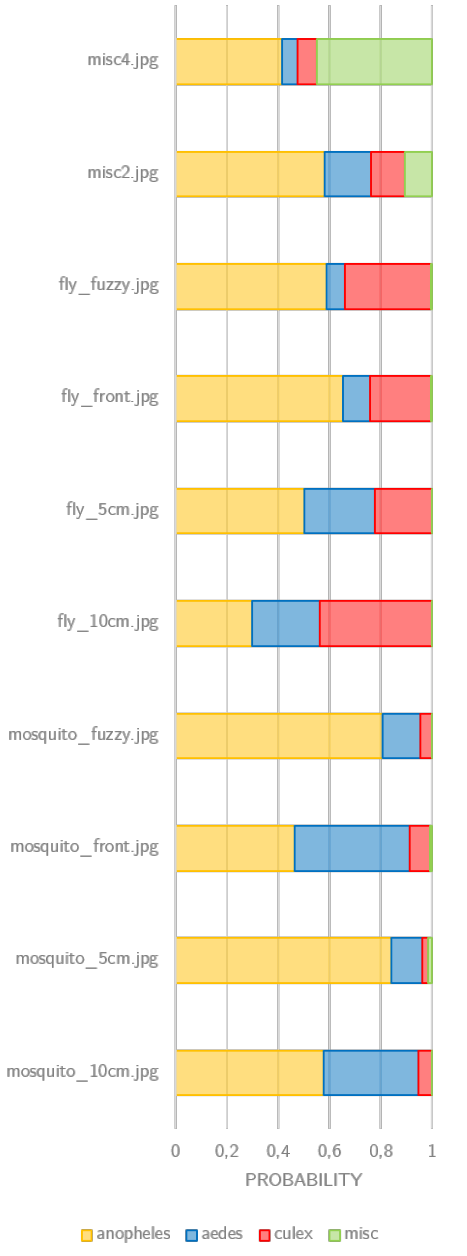
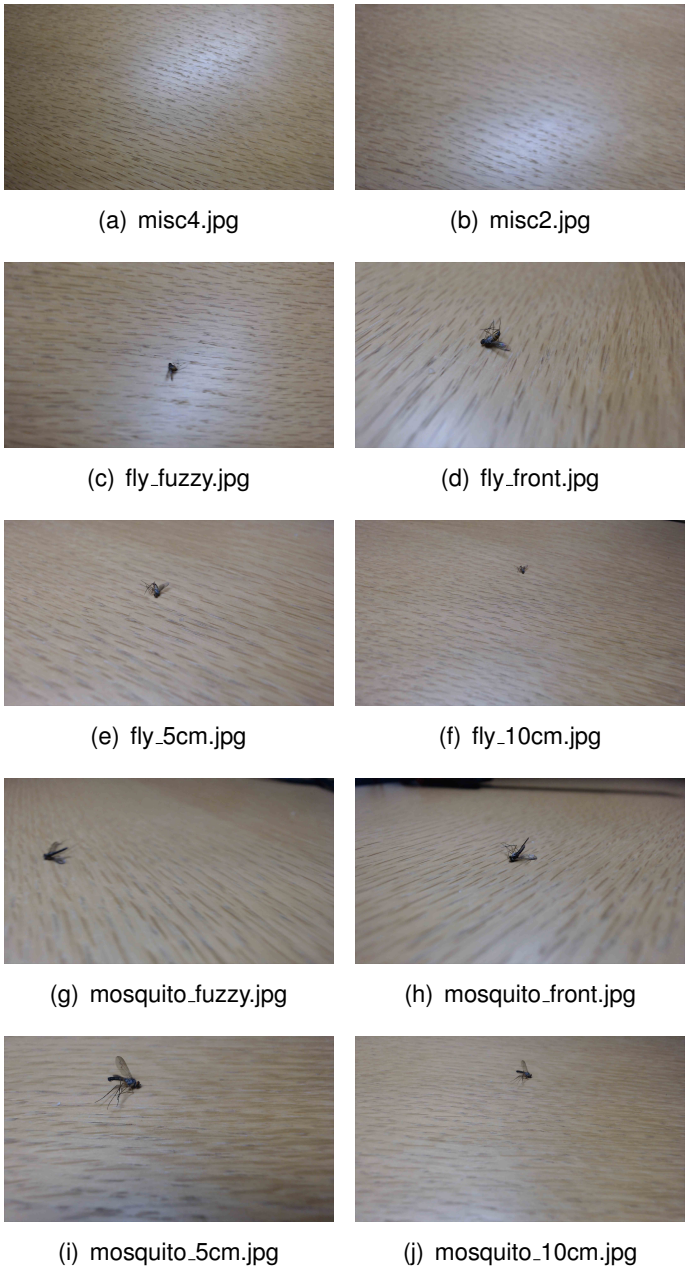


Figure 6.8: Probabilities distribution



(a) misc4.jpg (b) misc2.jpg (c) fly_fuzzy.jpg (d) fly_front.jpg (e) fly_5cm.jpg (f) fly_10cm.jpg (g) mosquito_fuzzy.jpg (h) mosquito_front.jpg (i) mosquito_5cm.jpg (j) mosquito_10cm.jpg

Figure 6.9: Pictures taken with *Logitech* C920 HD Pro

Some of the taken pictures and their corresponding probability are shown in Figure 6.8 and 6.9. There are 10 pictures, 4 of a mosquito, 4 of a fly and 2 of the background into which the insects were photographed. Moreover, the insects have been photographed from 10 cm and 5 cm, from the front and in fuzzy conditions. Notice that, in addition to a mosquito, a fly has also been photographed as it is the most similar insect to a mosquito, which will

test how well the mosquito classification is done by the algorithm.

Such as Figure 6.8 demonstrates, the mosquito classification is correctly effectuated in 2 out of 4 images of the mosquito. In fact, for ”mosquito_5cm.jpg” and ”mosquito_fuzzy.jpg”, the probability of containing an anopheles atroparvus mosquito is around 0.8. Nevertheless, when classifying the pictures named ”mosquito_10cm.jpg” and ”mosquito_front.jpg”, the CNN doubts mainly between two labels which are ”anopheles” and ”aedes”. In these previous cases, the algorithm knows that there is a mosquito in the shown images but it does not determine the type accurately.

Regarding the pictures of the fly, for the ones called ”fly_10cm.jpg” and ”fly_5cm.jpg”, the CNN hesitates between the three categories which it was trained for, with more or less the same probability. This means that it knows that there is an insect in these images, but it does not know to which label it belongs. For the other two remaining pictures of the fly, although the algorithm doubts between ”anopheles” and ”culex” labels, the probability of being classified such an ”anopheles” is much higher. That is why, in this cases, the classification can be considered failed.

Finally, for the picture named ”misc4.jpg”, the highest probability is the one corresponding to ”misc” label (with a probability of almost 0.5). Thus, in this case, the classification is successfully effectuated. However, for the one called ”misc2.jpg”, although the highest probability is not assigned to ”misc” label, the probability given to this label, which is 0.1 approximately, is much higher than for the pictures containing insects.

Although these pictures, representing the different results obtained after processing the taken pictures in Barcelona, are few, they are considered enough to prove the proper functioning of the chosen camera in controlled spaces. Hence, if it is kept in mind that for mosquito detection, the final test accuracy is about 69%, it can be said that the camera verification for mosquitoes detection has been carried out successfully.

6.3. Ground system validation

To assess the designed ground system, the water temperature sensor is going to be calibrated. In addition, the accuracy of the water temperature sensor as well as the one of the water depth sensor are going to be determined.

6.3.1. Water temperature sensor calibration

As exposed in Section 5.4.1., from the resistor value of the water temperature sensor, the temperature is computed. Nevertheless, when applying directly the theoretical equation (see Equation 5.1) converting a resistor value into a temperature value, the obtained temperature value is not real. Therefore, to solve this mismatch of values, a sensor calibration is required.

To achieve it, an empty plastic bottle has been cut and fill with a certain quantity of water, which has not being changed during the calibration process. Then, such plastic bottle has been exposed to sunlight in repeated times but during different amounts of time. After each sunlight exposure, the temperature value given by the water temperature sensor, and the one given by the handheld Multi 340i multimeter (which measures the real temperature

from the water capacity), have been collected.

The result obtained after different sunlight exposures can be seen in Figure 6.10 where 7 different values of temperature have been taken. Indeed, from the relation between the real temperature and the one given by the sensor, a regression line can be plotted whose equation (see Equation 6.1) corrects Equation 5.1, calibrating hence, the sensor.

$$T_{real} = 1.0477 T_{sensor} - 14.499 \tag{6.1}$$

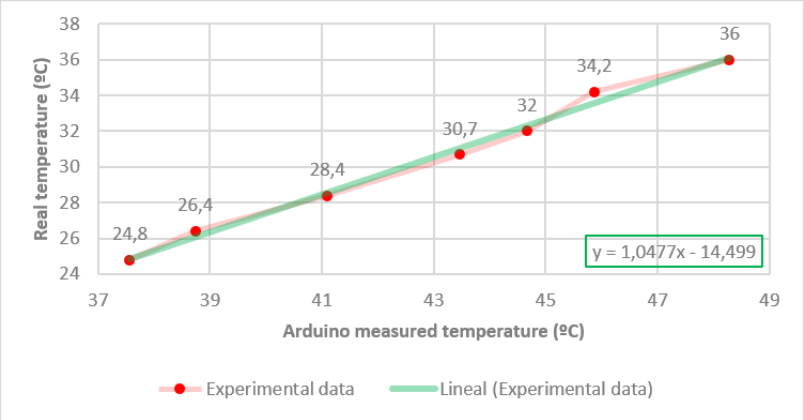
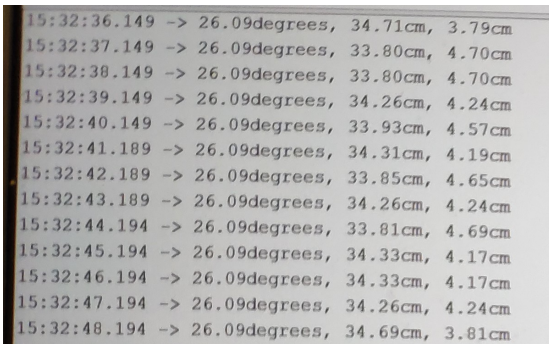


Figure 6.10: Relationship between the real water temperature and the one measured by the water temperature sensor

Finally, applying this correction to the script called ”groundMeasurements.ino” (see Appendix K), the water temperature given is finally realistic. In fact, Figure 6.11 proves it, as the temperature given by the handheld Multi 340i multimeter, which is 26.0 ℃, and the one given by the water temperature system, which is 26.09 ℃, are almost equal.

(a) Water temperature reading using Multi 340i multimeter



(b) Water temperature reading using the tested sensor

Figure 6.11: Matching systems after water temperature sensor calibration

6.3.2. Ground system accuracy

To determine the accuracy of the ground system, 50 samples of water temperature and water depth has been taken to then, plot the normal distribution of such samples. It should

be noted that these samples have been taken using the same bottle with the same water volume than Section 6.3.1..

6.3.2.1. Water temperature

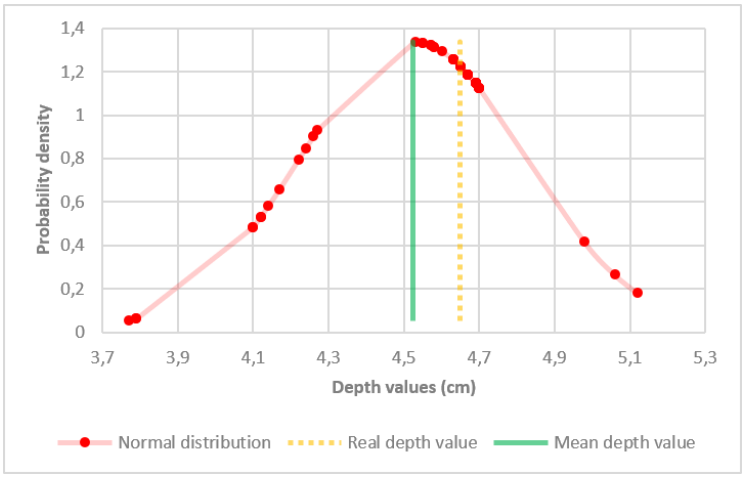
In the case of the water temperature, all the samples taken have the same value, which is 26.09 ℃. Consequently, plotting a normal distribution when the samples taken have the same value does not really make sense.

6.3.2.2. Water depth

In the case of the water depth, the samples taken have different values, allowing to plot their normal distribution.

To achieve it, the mean value of the samples have been computed as well as the standard deviation. In addition, the real water depth has been measured with a ruler to be able to compare it with the sensor measurements (see Fig. 6.12(b)). The resulting values of these parameters can be observed in Table 6.3.2.2.. Hence, using these values and *Microsoft* Excel, the normal distribution is obtained such as in Figure 6.12(a).

Notice that in Figure 6.12(a), the normal distribution is plotted in red, the mean value with a green line and the real depth value with a yellow dashed line.



(a) Normal distribution of the water depth samples

(b) Measuring real water depth

Figure 6.12: Water depth accuracy assessment

Real water depth	4.65 cm
Mean	4.52 cm
Standard deviation (σ)	0.30 cm
Standard deviation (2σ)	0.60 cm
Bias	0.13 cm
Coefficient of variation	6.6%

Table 6.3: Normal distribution related parameters

From Figure 6.12(a), the bias and the coefficient of variation (CV) of the water depth sensor can be computed. In fact, the bias is the difference between the real depth value and the mean one (4.65 cm and 4.52 cm, respectively), while the coefficient of variation is the quotient between one standard deviation and the mean value, given as a percent (0.3 cm and 4.52 cm, respectively). According to these definitions, the two last values of Table 6.3.2.2. are obtained.

Regarding the obtained bias value, it could be used to calibrate the system and to obtain a better estimation of the water depth. Nevertheless, the bias is so small compared to the depths to be measured that it has been considered to be not necessary. Furthermore, the lower the coefficient of variation is, the more precise the values are. This allows to say that the water depth sensor has a good precision and that it is enough to determine the puddles depth, whose values are of the order of magnitude of centimeters.

6.4. System validation

6.4.1. Validation for larval detection

To validate that the system is able to detect stagnant water from a picture, four pictures of the laboratory where the mosquito prevention system has been developed, have been taken and analyzed by the *Python* script called ”puddle_process.py” (see Appendix E for checking the code). Nevertheless, it should be noted that the system validation to detect mosquitoes has not been carried out as no mosquitoes were captured by the web-camera.

The first thing the script does, is to capture a picture such as the ones of Figure 6.4.1., and at the same time, saves the raw data given by the GPS (see Code 6.2) Then, this raw data is processed to extract the latitude and longitude and then, these coordinates are used to name the captured picture. For instance, if the latitude and longitude of a given picture are E1.987609 °and N41.275587 °, respectively, the name assigned to the picture is ”1987609E_41275587N.jpg”.

```
1 \$GPGGA,132326.00,4116.53522,N,00159.25659,E,1,06,2.27,28.1,M,51.4,M,,*60
2 \$GPGGA,132327.00,4116.53479,N,00159.25615,E,1,05,2.34,27.5,M,51.4,M,,*6C
3 \$GPGGA,132328.00,4116.53475,N,00159.25601,E,1,05,2.33,27.7,M,51.4,M,,*6F
4 \$GPGGA,132329.00,4116.53467,N,00159.25577,E,1,07,1.77,27.8,M,51.4,M,,*61
```

Listing 6.2: ”Raw data given by the GPS”

(a) ”1987609E_41275587N.jpg”

(b) ”1987602E_41275579N.jpg”

(c) ”1987600E_41275579N.jpg”

(d) ”1987596E_41275577N.jpg”

Figure 6.13: Pictures taken by the camera

After saving the GPS raw data in a file, the script captures the humidity, the temperature and the temperature index of the air given by the air temperature and humidity sensor. Next, these environmental and positioning data, associated with each captured picture, are gathered in the file called ”airMeasurements.csv” such as Table 6.4 and 6.5 show. However, these environmental data are purely informative, as for puddle detection, what it is fundamental is to know the puddle conditions, that means the stagnant water temperature and depth.

Picture	Datetime	Latitude	Longitude
1987609E_41275587N.jpg	03/07/2019 15:24:42	41275587N	1987609E
1987602E_41275579N.jpg	03/07/2019 15:25:30	41275579N	1987602E
1987600E_41275579N.jpg	03/07/2019 15:26:21	41275579N	1987600E
1987596E_41275577N.jpg	03/07/2019 15:27:08	41275577N	1987596E

Table 6.4: First part of the content of the file ”airMeasurements.csv”

Picture	Humidity [%]	Temp [℃]	Temp index [℃]
1987609E_41275587N.jpg	47.0	26.0	25.88
1987602E_41275579N.jpg	46.0	26.0	25.86
1987600E_41275579N.jpg	47.0	26.0	25.88
1987596E_41275577N.jpg	47.0	26.0	25.88

Table 6.5: Second part of the content of the file ”airMeasurements.csv”

Then, once the obtained air sensor data, the captured picture is processed by the CNN trained for puddle recognition. As a result of the CNN classification, the probability for such picture of containing a puddle and the probability of being a miscellaneous picture, are obtained. As four pictures have been captured for this validation, Figure 6.14 presents the results of such probability distribution for such pictures. After analyzing puddle pictures downloaded from the Internet and the ones taken with the chosen camera, as in Section 4.5.2. and 6.2.3., respectively, it has been concluded that, when the CNN analyzes a picture with a puddle, the probability obtained for the label ”puddle” is normally higher

than 0.85. Thus, it has been decided that, if the probability of the label ”puddle” is not higher than 0.85, the final label assigned must be ”misc”. This explains why the final label assigned for the four assessed pictures is ”misc” (see Fig. 6.6), and not ”puddle” for the one named ”1987600E_41275579N.jpg”, although the label ”puddle” has a higher probability than the label ”misc”.

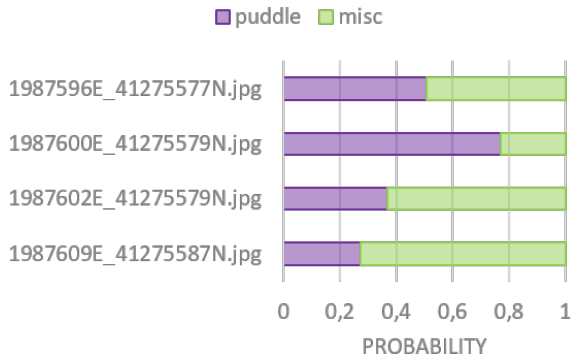


Figure 6.14: Results after processing the pictures taken

Picture	Label
1987609E_41275587N.jpg	misc
1987602E_41275579N.jpg	misc
1987600E_41275579N.jpg	misc
1987596E_41275577N.jpg	misc

Table 6.6: First part of the content of the file ”airMeasurements.csv”

After determining if the taken pictures contains a puddle or not, and, knowing the GPS coordinates of the pictures taken, the detected puddles can be plotted such as in Figure 6.15. The script called ”draw_sizepin_from_file.py” (see Appendix G for more details of the code), is in charge of generating kml files to then open them from Google Earth.

Figure 6.15: Location of the pictures taken

At the same time pictures are taken and processed, through the use of the *Python* script called ”save_bluetooth_data.py” (see Appendix H for more details of the code), when the air system is close enough to the ground system located in a given puddle, the ground data is transmitted from the ground system to the air system. An example of the transmitted data is Code 6.3.

```
03/07/2019 15:15:06  24.87 degrees, 34.24 cm, 4.26 cm
03/07/2019 15:15:07  24.87 degrees, 33.90 cm, 4.60 cm
03/07/2019 15:15:08  24.87 degrees, 33.90 cm, 4.60 cm
03/07/2019 15:15:09  24.87 degrees, 33.90 cm, 4.60 cm
03/07/2019 15:15:10  24.87 degrees, 34.74 cm, 3.76 cm
03/07/2019 15:15:11  24.87 degrees, 33.95 cm, 4.55 cm
03/07/2019 15:15:12  24.87 degrees, 34.30 cm, 4.20 cm
03/07/2019 15:15:13  23.64 degrees, 34.31 cm, 4.19 cm
03/07/2019 15:15:14  23.64 degrees, 33.88 cm, 4.62 cm
03/07/2019 15:15:15  23.64 degrees, 34.31 cm, 4.19 cm
```

Listing 6.3: ”Raw data measured by the ground system and received through Bluetooth”

Finally, it is only about modifying, if necessary, the labels assigned to the processed pictures taking into account the ground measurements to increase the probability of successfully detecting puddles containing mosquito larvae. This means retrieving each picture in which a puddle was detected (i.e. the ones with the assigned label ”puddle”), with its corresponding date and time and, assessing the ground measurements collected within an interval of $\pm$10 minutes from the moment such picture was taken. Evaluating the ground measurements within this interval allows to be quite sure that the collected ground data are related to the puddle captured by the camera. In fact, this interval should be reduced in order to associate ground and air data with more precision. Hence, if the gathered ground data do not fulfill the conditions for larval growth, the label of such picture is switched to ”misc”.

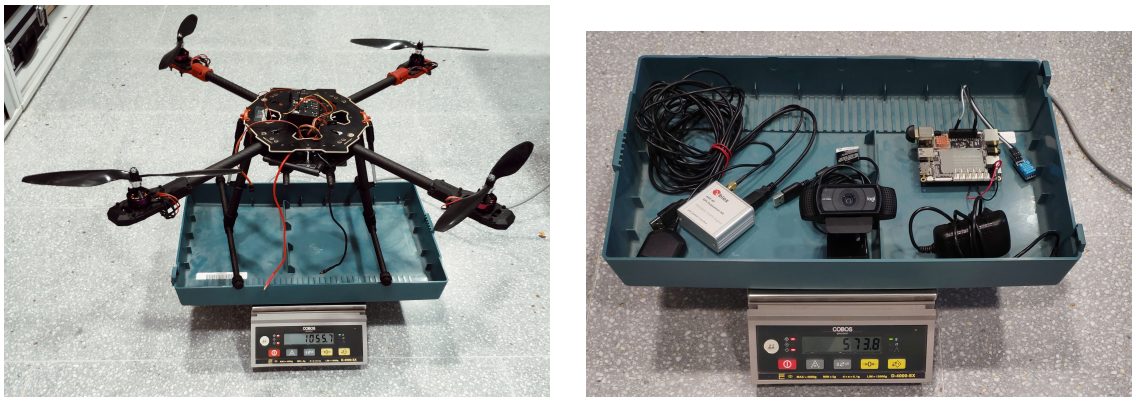
Notice that this post processing is executed by the *Python* script called ”puddle_and_btb_data_comparison.py” (see Appendix I for more details of the code), but as no puddles were detected during the validation, it has not been validated.

To sum up, it can be affirmed the designed system is validated for puddle detection, according to the given system specifications.

6.4.2. Proposition of system support structure

To end up with the validation of the mosquito prevention system, next, it is going to be proved that it can be easily fitted in a drone. It is only a matter of weight adjustment and battery life.

To achieve it, the total weight that the drone rotors should lift, has to be determined. In this way, the drone structure and the mosquito prevention hardware, excluding the ground hardware, have been weight as Figure 6.4.2. shows. Indeed, the drone structure has a weight of 1055.7 g, while the hardware has a weight of 573.8 g. If these weights are summed, the total weight is about 1629.5 g.



(a) Weight of the drone structure

(b) Weight of the used hardware

Figure 6.16: Determining the weights of the elements to integrate

However, this total weight is not the final weight that the drone rotors should lift, as the battery weight has still to be added. Let’s take Li-Po MULTISTAR 3S 11.1 V 5200 mAh 10C as the battery model for the selected drone structure (see Ref. [79] for more information about the selected battery). In fact, this battery is characterized by having 3 cells with 11.1

V and a capacity of 5200 mAh keeping a light weight, which is of 331 g. Hence, the total weight the drone rotors should lift is 1960.5 g.

As the drone structure used is the one of a quad-copter, the weight lifted by each of their rotors is a quarter of the total weight, that means 490.1 g. In order to determine the rotor features such that it is able to lift such weight, the web tool called "Estimate Propeller's Static Thrust" from https://rcplanes.online/calc_thrust.htm has been used, such as Figure 6.17 show. The data used to obtain the final rotor features, are indicated in Table 6.7. To set these data, it has been assumed that the total duration of a flight is about 30 minutes, that the drone does not fly more than 10 meters high and that the flight is performed with an ambient temperature of 25 ℃.

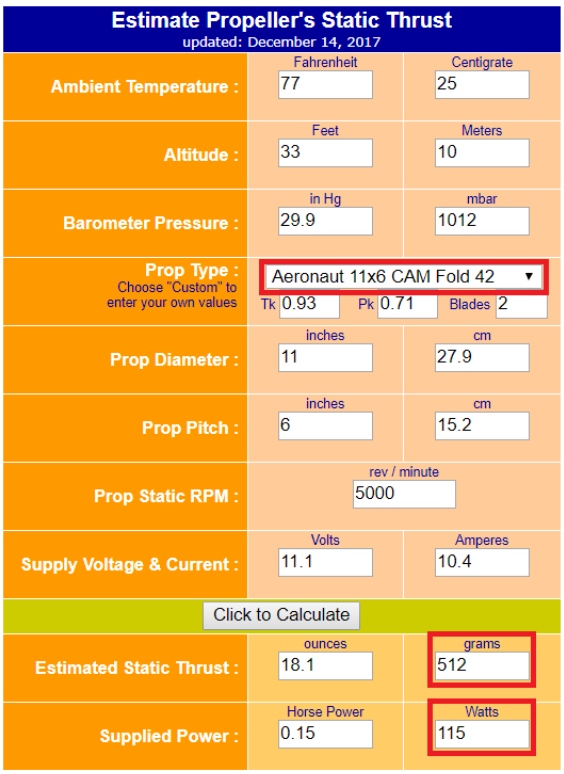


Figure 6.17: Determined rotor features

Ambient temperature	25 ℃
Max height	10 m
Weight per rotor	490.1 g
Voltage	11,1 V
Battery capacity	5.2 Ah
Current for 30 min	10.4 A

Table 6.7: Used data to determine the rotor features

From Figure 6.17, it can be established that the rotor needed for the designed system, has to be supplied with 115 W and it has to carry the *Aeronaut* 11x6 CAM fold 42 blade model. Notice these blades can generated a thrust of 512 g, which is enough for such system, as the minim required per rotor is 490.1 g.

To conclude, it can be affirmed the designed system can be fitted without no problem in a drone.

CHAPTER 7. BUDGET

This short chapter exposes the price of developing the whole project, taking into account all the elements already mentioned in Section 1.2., 3.2., 3.3., 5.4. and 5.5..

The cost of the whole project is summarized in Table 7 according to one of the aims of this project which is to make it as cheapest as possible in order to be affordable for the maximum number of people (specially the diagnostic part).

Notice that the price of implementing the first part of the project (i.e. the diagnostic perspective) is the price of its development minus the *ZugMed* Multi-Parameters Module Software price, that is 615.03 €. In fact, this software was paid once at the beginning of the project to get the corresponding code and thereby to develop the back-end program of the MMP system. Therefore, it is not necessary anymore to pay again for it.

Regarding the implementation of the second part of the project (i.e. the prevention perspective), the price of all the used elements sums up to a total of 556.97 €.

Thus, the whole project has a price of 1395.14 €. Keep in mind this price is the cost of all the elements used, without taking into account the price of the development made by an engineer. Yet, the production price of this system would be much less than the one of the development, making it even more affordable.

Article	**Original price**	**Price in €**
Diagnostic perspective		
ZugMed Multi-Parameters Module	408.00 $	364.03
ZugMed Multi-Parameters Module Software	250.00 $	223.14
ABS Hammond 1599HBK Box [22]	10.90 €	10.90
Microsoft Surface Pro (Model 1514) [23]	209.77 $	184.11
Huawei E8372 [29]	45.99 €	45.99
3G/4G sim card (*Pepephone* company) [80]	10.00 €	10.00
Prevention perspective		
LattePanda 4G/64GB [81]	209.00 $	183.40
u-blox AEK-4P (discontinued, similar to EVK-7) [82]	220.10 €	220.10
Logitech C920 Pro HD [58]	67.99 €	67.99
DHT11 [77]	9.93 €	9.93
Arduino Uno Rev3 [83]	20.00 €	20.00
Arduino XBee Shield Rev3 [76]	30.99 $	27.19
RS PRO 3 wire PT100 Sensor [73]	£21.48	24.01
HC-SR04 [75]	4.35 €	4.35
TOTAL		1395.14 €

Table 7.1: Final budget

CONCLUSIONS

This project has been carry out always thinking of how to bring something new to society and, in this way, how to improve somehow the way humans live. Thus, this project has been focused on developing two new solutions to combat a deadly disease such as malaria. Indeed, if these two solutions are combined, they form a double-edge tool from which the fight against any disease caused by any flying insect is much more powerful.

When developing such systems, three premises have been always kept in mind: the systems had to be the cheapest possible, ready to be used in all over the world and accessible by the greatest possible number of people, without forgetting the respect for the ecosystem. This explains the effort put in designing these systems and discussing the selected devices.

Once the systems have been tested, verified and validates, let's come back to the initial objectives to assess if they have been successfully accomplished and in which way.

Starting with the multiparametric system, the objective of designing of a multiparametric system able to measure, view, save and recover the principal vital constants of a human-being, have been achieved successfully after being tested with a real patient. In addition, the transmission of all the data generated by the multiparametric system to any place of the world, can be considered to be accomplished as it only depends on the use of a 3G/4G module, which was successfully tested with the developed system. Moreover, these objectives have been fulfilled integrating the already mentioned premises, which means the system has being designed to be portable, easy-to-use, fast and cheap as well as available for all world's inhabitants.

In this way, these two main purposes give to the multiparametric system the potential to be used anywhere in the world, no matter where the patient and the hospital with medical professionals are located. The only requirement needed is to have GSM, 3G or 4G coverage. This fact gives the possibility to the patient, in case of suffering from a heavy disease, to be monitored by the best medical professionals, without the need of being next to the patient to make a diagnostic. At the same time, this leads to a time saving, as there is no need to move from a seat, and to the possibility that medical professional could treat more patients.

Regarding the mosquito prevention system, the objective of automatically detecting a puddle from a picture through the use of Artificial Intelligence, has been successfully carried out with a really high probability of detecting correctly the presence of water. However, it is not the case when trying to automatically identify different mosquitoes species from a picture through the use of Artificial Intelligence. Related with this objective, it has to be highlighted that the automated detection of mosquitoes have not been tested in real conditions as the pictures taken to validate the detection of the system were taken in the laboratory without mosquitoes and not in mosquito natural habitat. However, when using Internet images, it can be said that the mosquito detection has worked rather well, if it is reminded the final test accuracy obtained by the used algorithm. In terms of the design of a sensors system able to gather environmental data from the ground and from the air, it can be affirmed that it was achieved as expected as these sensors have been validated and their data have been successfully collected.

Finally, the objective of integrating the sensors system and the automated recognition sys-

tem of mosquitoes and puddles in a single architecture, can be considered to be accomplished. Without any doubt, it is the most important aim of the mosquito prevention system, as the resulting multi-information architecture has the potential to enhance the probabilities given by Artificial Intelligence. In fact, by mixing Artificial Intelligence with real environmental data, the number of correct detections could improve incredibly, highlighting hence, the feasibility of such architecture. In addition, although, at the beginning of the project, it was not one of the objectives to reach, the feasibility of the integration of the system in a drone, has been proved.

Similarly to the first system, these objectives have been fulfilled integrating the already mentioned premises, which means the mosquito prevention system has being designed to be fast, effective, localized, cheap and respectful with the ecosystem. At the same time, this proves that such system could be used anywhere in the world either to fight against mosquitoes, or, at the end, to monitor any flying insect.

Now, the architecture of the designed systems is clearly exposed allowing anyone to implement such systems in any place of the world with the hardware of his or her choice.

Future work

Up to now, two system prototypes have been developed and their feasibility have been assessed. Actually, this is the beginning of two systems that could help to improve the quality of life for lots of people. However, there is still a considerable amount of tasks to carry out and details to refine before they could be used by the general public.

In this way, the remaining tasks to go ahead these prototypes, are listed below.

Concerning the multiparametric system, the tasks that should be carry out are the next ones.

- Improvement of the system performance to make the data transference between the back-end and the front-end faster. This means improving the performance of the socket used to communicate both parts.
- Modification of the hardware used in such a way it could certified as an IP67 equipment and thus, it could be used anywhere in the world. This can be translated to the selection of components protecting such hardware from heat, cold, rain, humidity and dust.
- Incorporation of measurements from external devices to the front-end in order to enhance the information given by the system.
- Testing of the system in real conditions with real patients and hospitals. In this project, the system has only be tested with two people.
- Incorporation of a voice or message system in order to be able to establish a conversation between patients and medical professionals and hence, to keep them in touch.

Regarding the mosquitoes prevention system, the tasks that should be carry out are the following ones.

- Enlargement of the number of pictures used to train CNNs. This means increasing especially the number of mosquitoes pictures in order to improve the probability of well classifying the new mosquitoes pictures shown to the AI algorithm.
- Enhancement of CNNs performance. This means adjusting their weights and parameters in order to improve the probability of well classifying the new pictures shown to the algorithms.
- Integration of the detection of puddles and mosquitoes in a single script in order to simplify the system. Up to the moment, the detection of puddles and mosquitoes has been split in two different scripts to evaluated them separately but once validated, this separation has no sense.
- Improvement of the image processing to enhance the mosquitoes density estimation per picture. Indeed and as already known, when a mosquitoes picture is taken, it is divided into smaller pictures and then each of them is processed. As a result of these divisions, a mosquito could appear partially in two different cut pictures and thus, it would be counted twice and could affect the density estimation.
- Improvement of the detection thresholds according to the probabilities given by the AI algorithm to improve the probability of labeling pictures correctly.
- Selection of more accurate and reliable sensors to estimate environmental conditions. In fact, the sensors used were not very accurate as what was sought is to design the cheapest system; nevertheless this is a matter of price (the highest the price, the more accurate the sensor).
- Combination of the automated detection of puddles and mosquitoes with other measurements of the environment as well as other features such as geographical location, distance to the sea and meteorological information, to identify patterns and generate a predictive model capable to anticipate the probability of presence of mosquitoes and their larvae, given certain conditions. This includes the extraction of the puddle model from the studied area.
- Integration of the system in a drone and a rover including the test of such system. Although it was pending to be done in this project, finally only the feasibility of the integration of the system in a drone has been demonstrated.

BIBLIOGRAPHY

[1] PM4100 Patient Monitor Modules. From: Berry [online]. BerryMedical, 2019. [Consultation: May 3rd, 2019]. Available on: <http://www.shberrymed.com/pm4100-patient-monitor-modules-p00039p1.html>. 8

[2] PM4100 Patient Monitor Modules. From: Alibaba.com [online]. Alibaba.com, 2019. [Consultation: May 3rd, 2019]. Available on: <https://www.alibaba.com/product-detail/spo2-oem-module-6750-Patient-Monitor_528875453.html?spm=a2700.7724838.2017115.1.6c5959b8fjXSxa&s=p>. 8

[3] (r)evolution Board Kit BT. From: bitalino [online]. BITalino, 2019. [Consultation: May 3rd, 2019]. Available on: <https://bitalino.com/en/board-kit-bt>. 8

[4] ChipOx Pulse Oximetry Board. From: corscience [online]. Corscience & GmbH & Co, 2018. [Consultation: May 3rd, 2019]. Available on: <https://www.corscience.com/chipox>. 8

[5] COR12: 12 Channel Bluetooth ECG. From: corscience [online]. Corscience & GmbH & Co, 2018. [Consultation: May 3rd, 2019]. Available on: <https://www.corscience.com/cor12>. 8

[6] OEM Solutions. From: Masimo [online]. Masimo, 2018. [Consultation: May 3rd, 2019]. Available on: <http://www.masimo.com/oem/solutions/>. 8

[7] MX-3 Masimo Rainbow Set OEM Board. From: Dotmed [online]. DOTmeda.com, 2018. [Consultation: May 3rd, 2019]. Available on: <https://es.dotmed.com/listing/other/masimo/mx-3-rainbow-set-oem-board/23376/1871707>. 8

[8] Multiparameter Modules. From: medlab [online]. medlab medizinische Diagnosegeräte GmbH, 2019. [Consultation: May 3rd, 2019]. Available on: <http://www.medlab-gmbh.de/english/modules/multiparameter-monitoring/index.html>. 8

[9] Nellcor™Multi-functional Respiratory PCBA OEM Platform. From: Medtronic [online]. Medtronic, 2019. [Consultation: May 3rd, 2019]. Available on: <https://www.medtronic.com/covidien/en-us/products/oem-monitoring-solutions/nellcor-pulse-oximetry/nellcor-multi-functional-respiratory-pcba-oem-platform.html>. 8

[10] Medtronic Covidien Multi Parameter PM PCBA-1. From: cpapusa [online]. cpapusa.com, 2019. [Consultation: May 3rd, 2019]. Available on: <https://www.cpapusa.com/multi-parameter-pm-pcba-1.html>. 8

[11] NIBP Development Kit. From: PAR Medizintechnik [online]. PAR MEDIZINTECHNIK GMBH & CO, 2019. [Consultation: May 3rd, 2019]. Available on: <http://www.par-berlin.com/en/products/nibp-development-kit/>. 8

[12] PAR Port. From: PAR Medizintechnik [online]. PAR MEDIZINTECHNIK GMBH & CO, 2019. [Consultation: May 3rd, 2019]. Available on: <http://www.par-berlin.com/en/products/par-port/>. 8

[13] OEM Electrocardiography Module. From: VectraCor [online]. VectraCor, 2019. [Consultation: May 3rd, 2019]. Available on: <https://www.vectracor.com/oem-electrocardiography-module/>. 8

[14] OEM Modules. From: ZugMed [online]. ZugMed Medical Systems SAS, 2018. [Consultation: May 3rd, 2019]. Available on: <https://www.zugmed.com/oem-modules/multi-parameters-module/>. 7, 8

[15] ECG Lead Positioning. From: Life in the fast lane [online]. LITFL, 2018. [Consultation: May 3rd, 2019]. Available on: <https://litfl.com/ecg-lead-positioning/>. 13

[16] ZUG Medical Systems SAS. *The Developer manual of Integrated Multi-Parameter Module*. (ZUG Medical Systems SAS. Sophia Antipolis. 2018): Internal Document. 13, 14, 15, 17, 18

[17] Code for fast integer to string conversion in *C++*. From: Ideone [online]. Sphere Research Labs., 2019. [Consultation: April 27th, 2019]. Available on: <https://ideone.com/nrQfA8>. 25

[18] Fast integer to string conversion in *C++*. From: Zverovich Blog [online]. Victor Zverovich, 2019. [Consultation: April 27th, 2019]. Available on: <http://www.zverovich.net/2013/09/07/integer-to-string-conversion-in-cplusplus.html>. 25

[19] What is MySQL?. From: MySQL [online]. Oracle Corporation and/or its affiliates, 2019. [Consultation: April 25th, 2019]. Available on: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>. 26

[20] Travis, J. "Client-Server Applications with LabVIEW". *Internet Applications in LabVIEW*. (Prentice-Hall. New Jersey. 2000): 55–77. 39

[21] LabView Database Library. From: SourceForge [online]. Slashdot Media, 2019. [Consultation: April 3rd, 2019]. Available on: <https://sourceforge.net/projects/sql-lv/>. 40

[22] Caja de ABS Hammond 1599HBK. From: RS Components [online]. RS, 2019. [Consultation: May 23th, 2019]. Available on: <https://es.rs-online.com/web/p/cajas-de-uso-general/2072091/>. 41, 93

[23] Microsoft Surface Pro Tablet 128 GB Hard Drive, 4 GB RAM, Windows 8 Pro. From: Amazon [online]. Amazon.com, 2019. [Consultation: June 25th, 2019]. Available on: <https://www.amazon.com/gp/offer-listing/B00BE5T2TA/ref=dp_olp_0?ie=UTF8&condition=all&qid=1561471062&sr=1-9>. xv, 42, 93

[24] 4G LTE World Coverage Map. From: WorldTimeZone [online]. WorldTimeZone, 2019. [Consultation: April 3rd, 2019]. Available on: <https://www.worldtimezone.com/4g.html>. xv, 43

[25] Docooler 4G Portátil Mini WiFi Router USB Modem. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Docooler-Port%C3%A1til-Router-100Mbps-Tarjeta/dp/B07PM6PWVC>. 43

[26] D-Link DWM-222. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/D-Link-DWM-222-Cualquier-Operador-Compatible/dp/B00PVDQ37A>. 43

[27] Huawei E3372. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Huawei-E3372-Adaptador-Color-Blanco/dp/B0104LV06M>. 43

[28] Huawei E3531i-2. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Huawei-Technology-Ltd-E3531i-2-21-6Mbps/dp/B011YZZ6Q2>. 43

[29] Huawei E8372. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Huawei-E8372-Modem-WiFi-LTE/dp/B014WMEJ2Q>. xv, 43, 93

[30] Vital Signs. From: Cleveland Clinic [online]. Cleveland Clinic, 2019. [Consultation: June 27th, 2019]. Available on: <https://my.clevelandclinic.org/health/articles/10881-vital-signs>. 46, 48

[31] Temperature Of A Healthy Human (Skin Temperature). From: The Physics Factbook [online]. Hypertextbook, 2019. [Consultation: June 27th, 2019]. Available on: <https://hypertextbook.com/facts/2001/AbantyFarzana.shtml>. 46, 48

[32] What your blood oxygen level shows. From: Healthline [online]. Healthline Media, 2019. [Consultation: June 27th, 2019]. Available on: <https://www.healthline.com/health/normal-blood-oxygen-level#oxygen-levels>. 46, 48

[33] What is Perfusion Index (PI). From: AmperorDirect.com [online]. Amperor, Inc., 2019. [Consultation: June 27th, 2019]. Available on: <https://www.amperordirect.com/pc/help-pulse-oximeter/z-what-is-pi.html>. 46, 48

[34] Goodfellow, I., Bengio, Y., Courville A. *Deep Learning*. (MIT Press. New Jersey. 2016). [Consultation: May 25th, 2019]. Available on: <http://www.deeplearningbook.org>. 49, 54

[35] Advanced Guide to Inception v3 on Cloud TPU. From: Google Cloud [online]. Google Cloud, 2019. [Consultation: May 16th, 2019]. Available on: <https://cloud.google.com/tpu/docs/inception-v3-advanced>. 49, 50

[36] Inception V3 Deep Convolutional Architecture For Classifying Acute Myeloid/Lymphoblastic Leukemia. From: Intel AI Developer Program [online]. Intel Software Developer Zone, 2019. [Consultation: May 16th, 2019]. Available on: <https://software.intel.com/en-us/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic>. 49

[37] Szegedy, C. Vanhoucke, V. Ioffe, S. Shlens, J. Wojna,Z. “Rethinking the Inception Architecture for Computer Vision”(2015) 49

[38] Inception v3. From: MathWorks [online]. The MathWorks, Inc, 2019. [Consultation: May 16th, 2019]. Available on: <https://es.mathworks.com/help/deeplearning/ref/inceptionv3.html>. 50

[39] Image Classification Transfer Learning with Inception v3. From: Google Developers Codelabs [online]. Google Developers, 2019. [Consultation: May 16th, 2019]. Available on: <https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/index.html#1>. xv, 50

[40] Convolutional Neural Networks for Beginners. From: Towards Data Science [online]. Towards Data Science, 2019. [Consultation: May 25th, 2019]. Available on: <https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca>. 50

[41] Géron, A. "Logistic Regression". *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. (O'Reilly Media, Inc. Sebastopol. 2017): 141–144. 50

[42] Tipos de mosquitos. From: Rentokil [online]. Rentokil Initial plc, 2019. [Consultation: May 21th, 2019]. Available on: <https://www.rentokil.es/mosquitos/tipos-de-mosquitos/>. 51

[43] How to Retrain an Image Classifier for New Categories. From: TensorFlow [online]. TensorFlow Developers, 2019. [Consultation: May 19th, 2019]. Available on: <https://www.tensorflow.org/hub/tutorials/image_retraining>. 53, 54

[44] Géron, A. "Performance measures". *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. (O'Reilly Media, Inc. Sebastopol. 2017): 85–86. 54

[45] Deep Learning with Tensorflow: Part 2 - Image classification. From: Towards Data Science [online]. Towards Data Science, 2019. [Consultation: May 19th, 2019]. Available on: <https://towardsdatascience.com/deep-learning-with-tensorflow-part-2-image-classification-58fcdffa7b84>. 55

[46] "classify.py" code from *tensorflow-image-classifier*. From: GitHub [online]. GitHub, Inc., 2019. [Consultation: May 21th, 2019]. Available on: <https://github.com/burliEnterprises/tensorflow-image-classifier/blob/master/classify.py>. 53

[47] Note on Difference between Culex and Anopheles Mosquitoes. From: Kullabs [online]. Kullabs, 2019. [Consultation: May 20th, 2019]. Available on: <https://www.kullabs.com/classes/subjects/units/lessons/notes/note-detail/810>. 58

[48] Aedes albopictus. From: Wikipedia [online]. Fundación Wikimedia, Inc., 2019. [Consultation: May 22th, 2019]. Available on: <https://es.wikipedia.org/wiki/Aedes_albopictus>. 74

[49] Anopheles. From: Wikipedia [online]. Fundación Wikimedia, Inc., 2019. [Consultation: May 22th, 2019]. Available on: <https://en.wikipedia.org/wiki/Anopheles>.

[50] Anopheles. From: Wikipedia [online]. Fundación Wikimedia, Inc., 2019. [Consultation: May 22th, 2019]. Available on: <https://en.wikipedia.org/wiki/Anopheles>. 74

[51] Jones, R. "ANOPHELINE MOSQUITOES, Anopheles and other species". *House Guests, House Pests: A Natural History of Animals in the Home*. (Bloomsbury Publishing Plc. London. 2015): 246. 74

[52] Culex. From: Wikipedia [online]. Fundación Wikimedia, Inc., 2019. [Consultation: May 22th, 2019]. Available on: <https://es.wikipedia.org/wiki/Culex>. 74

[53] Aukey PC-LM1-EU. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/AUKEY-Micr%C3%B3fono-Est%C3%A9reo-Grabaci%C3%B3n-Compatible/dp/B0721MKXQ2>. 74

[54] Besteker 920C-001. From: Amazon.com [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.com/Webcam-Besteker-Microphones-Widescreen-Facial-enhancement/dp/B073NZZF7R/ref=cm_cr_arp_d_product_top?ie=UTF8>. 74

[55] Intel Real Sense Depth Camera D435. From: Intel [online]. Intel Corporation, 2019. [Consultation: May 23th, 2019]. Available on: <https://click.intel.com/intelr-realsensetm-depth-camera-d435.html>. 74

[56] Logitech Brio. From: PCComponentes [online]. PC COMPONENTES Y MULTIMEDIA SLU,, 2019. [Consultation: May 23th, 2019]. Available on: <https://www.pccomponentes.com/logitech-brio-webcam-4k-ultrahd?gclid=EAIaIQobChMIos2DyOC84gIVD_lRCh3XhQecEAYYBCABEgK5J_D_BwE>. 74

[57] Logitech C615 HD. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Logitech-960-000735-C615-Webcam-negro/dp/B0050FBI4C>. 74

[58] Logitech C920 HD Pro. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Logitech-C920-HD-Pro-C%C3%A1mara/dp/B006A2Q81M>. 74, 93

[59] Logitech C922 Pro Stream. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Logitech-C922-Pro-Stream-Professional/dp/B01L6L52K4>. 74

[60] Microsoft LifeCam HD-3000. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.es/Microsoft-LifeCam-HD-3000-micr%C3%B3fono-integrado/dp/B0099XD1PU>. 74

[61] Razer Kiyo. From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <https://www.amazon.com/RAZER-KIYO-Adjustable-Autofocus-Streaming/dp/B075N1BYWB>. 74

[62] Kenyeres, Z. Bauer, N. Tóth, S. Sáringer-Kenyeres, T. “Habitat Requirements of Mosquito Larvae”, 5-6. (2011) 62

[63] Li, Y. Kamara, F. Zhou, G. Puthiyakunnon, S. Li, C. Liu,Y. Zhou, Y. Yao, L. Yan, G. Chen, X. “Urbanization Increases Aedes albopictus Larval Habitats and Accelerates Mosquito Development and Survivorship”. *PLOS*. **volum**(8), 7–10. (2014) 62

[64] Aedes albopictus From: Amazon.es [online]. Amazon.com, Inc., 2019. [Consultation: May 23th, 2019]. Available on: <http://www.europe-aliens.org/pdf/Aedes_albopictus.pdf>. 62

[65] Reiskind, M. Zarrabi, A. “Water Surface Area and Depth Determine Oviposition Choice in Aedes albopictus”. *Journal of Medical Entomology*. **volum**(49), 71–76. (2012) 62

[66] Ponce, G. Flores, A. Badii, M. Fernández, I. Rodríguez, M. “BIONOMÍA DE Aedes albopictus”. *Revista Salud Pública y Nutrición*. **volum**(5-2). (2004) 62

[67] Gao, Q. Wang, F. Lv, X. Cao, H. Su, F. et al. “Aedes albopictus production in urban stormwater catch basins and manhole chambers of downtown Shanghai, China”. *PLOS ONE*. **volum**(13). (2018) 62

[68] Brady, O et al. “A Modelling adult Aedes aegypti and Aedes albopictus survival at different temperatures in laboratory and field settings.”. *Parasites & Vectors*. **volum**(6). 351. (2013) 62

[69] Rowley, W. Graham, C. “The effect of temperature and relative humidity on the flight performance of female Aedes aegypti”. *Journal of Insect Physiology*. **volum**(14). 9. 1251-1257. (1968) 62

[70] Mordecai, E. et al. “Optimal temperature for malaria transmission is dramatically lower than previously predicted”. *Ecology Letters*. (2012) 62

[71] Umaru, N. Akogun, O. “Physical factors associated with Anopheles and Culex mosquitoes’ survival in captivity in Yola, Nigeria”. *International Journal of Modern Biological Research*. **volum**(4). 16-24. (2015) 62

[72] Lebl K. et al. “Predicting Culex pipiens/restuans population dynamics by interval lagged weather data.”. *Parasites & Vectors*. **volum**(6). 129. (2013) 62

[73] RS PRO 3 wire PT100 Sensor. From: RS Components [online]. RS Components Ltd., 2019. [Consultation: June 23th, 2019]. Available on: <https://export.rsdelivers.com/product/rs-pro/d00584-ps6-100-2000-1xpt100/rs-pro-3-wire-pt100-sensor-50c-min-200c-max-100mm/3730372>. 62, 93

[74] Ultrasonic Sensors: Applications in the Internet of Things. From: Seebo Blog [online]. Seebo Interactive LTD., 2019. [Consultation: July 1st, 2019]. Available on: <https://blog.seebo.com/iot-ultrasonic-sensors/>. xvi, 64

[75] Neuftech Module ultrasonido HC-SR04 From: Amazon [online]. Amazon.es, 2019. [Consultation: June 23th, 2019]. Available on: <https://www.amazon.es/Neuftech-ultrasonido-transductor-medici%C3%B3n-distancia/dp/B00PQB8GWM/ref=asc_df_B00PQB8GWM/?tag=googshopes-21&linkCode=df0&hvadid=546394218 75&hvpos=1o2&hvnetw=g&hvrand=10081892787845146 19&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1005424&hvtargid=pla-103789468566&psc=1>. 64, 93

[76] Arduino XBee Shield. From: Amazon [online]. Amazon.es, 2019. [Consultation: June 23th, 2019]. Available on: <https://www.amazon.com/gp/offer-listing/B004L6PNLA/ref=dp_olp_0?ie=UTF8&condition=all&qid=1561303933>. 65, 93

[77] DHT11. From: Amazon [online]. Amazon.com, 2019. [Consultation: June 23th, 2019]. Available on: <https://www.amazon.es/Temperatura-Humedad-Relativa-Sensor-Arduino/dp/B00HI7LUKW>. 67, 93

[78] Hardware Introduction. From: LattePanda Documentation [online]. LattePanda Team, 2019. [Consultation: July 4th, 2019]. Available on: <http://docs.lattepanda.com/content/1st_edition/hardware_introduction/>. 68

[79] Batería Li-Po MULTISTAR 3S 11.1V 5.200 mAh 10C. From: Modeltronic [online]. Modeltronic Motor S.L.U., 2019. [Consultation: July 7th, 2019]. Available on: <https://www.modeltronic.es/bateria-li-po-multistar-3s-111v-5200-mah-10c-para-multirotores-p-10637.html>. 90

[80] CUOTA MENSUAL. From: Pepephone [online]. Pepephone, 2019. [Consultation: June 26th, 2019]. Available on: <https://www.pepephone.com/movil/contratacion>. 93

[81] LattePanda 4G/64GB. From: DFROBOT [online]. dfrobot.com, 2019. [Consultation: June 23th, 2019]. Available on: <https://www.dfrobot.com/product-1585.html?search=LattePanda&description=true>. 93

[82] EVK-7 u-blox 7 GNSS evaluation kits. From: u-blox [online]. u-blox, 2019. [Consultation: June 24th, 2019]. Available on: <https://www.u-blox.com/en/product/evk-7>. 93

[83] Arduino Uno Rev3. From: Arduino [online]. Arduino, 2019. [Consultation: June 23th, 2019]. Available on: <https://store.arduino.cc/arduino-uno-rev3>. 93

APPENDICES

APPENDIX A. STEPS TO PROPERLY INSTALL THE MULTIPARAMETRIC SYSTEM

Taking into account that a multiparametric system is made of a back-end program, a front-end program and a database, Section A.1., A.2. and A.3. are intended to explain how to install these multiple parts and how to set them up.

A.1. Installing the back-end and the front-end software

As you might know, for the proper running of the multiparametric system, two different programs are needed: the one corresponding to the back-end part called *Multi_Para Monitor.exe* and the one corresponding to the front-end part named X.exe.

To get them working as expected in our own PC, they have to be downloaded from *github*. To accomplish that, go to `github.com/annatrial/Multi_Para_Monitor_x32/` and to `github.com/annatrial/MMP_LabView_x32`, respectively, and download from each link a zip file with the lastest version.

Once zip files have been decompressed, go to *Multi_Para-Monitor-V3* $\rightarrow$ *Debug* and check that the following dll files are present: *mfc140ud.dll*, *msvcp140d.dll*, *ucrtbased.dll*, *vcruntime140d.dll*, *msvcr90.dll*, *msvcp90.dll*, *mysqlcppconn.dll*. If it isn't the case, try to get them on the Internet and save them in this same folder.

A.2. Transforming the back-end program into a *Windows* service

According to the operating system used for this project, one of the possible solutions to avoid the user's manipulation of the back-end program, is to transform the generated exe file of this back-end program into a Windows service. To do so, some steps must be followed carefully discussed in the next Section.

A.2.1. Download of *nssm 2.24*

NSSM is a free service helper with which allows to easliy create a *Windows* service from any exe file. To download it, it is highly recommended to go to `nssm.cc/download` and look for the version 2.24. Once the zip file has been downloaded, extract the content into any directory of your PC.

A.2.2. Creation of a *Windows* service

The first step to create a Windows Service is to open a command prompt as administrator and copy and paste the next command to the cmd where *location folder* must be replaced by the folder in which *nssm-2.24* is placed.

```
Microsoft Windows [Version 10.0.17134.706]
Copyright (c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32> cd <location folder>\nssm-2.24\win64
```

Then, by typing the next command, where *Multi_Para Monitor* is the name given to our service, a window as the one in Figure A.1 will appear. The name given to this service must not be changed for the proper functioning of the entire system.

```
C:\Users\annat\Downloads\nssm-2.24\win64> nssm install "Multi_Para Monitor"
```

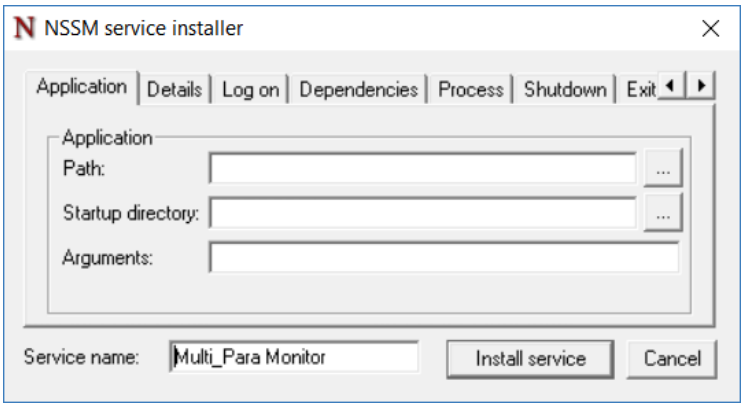
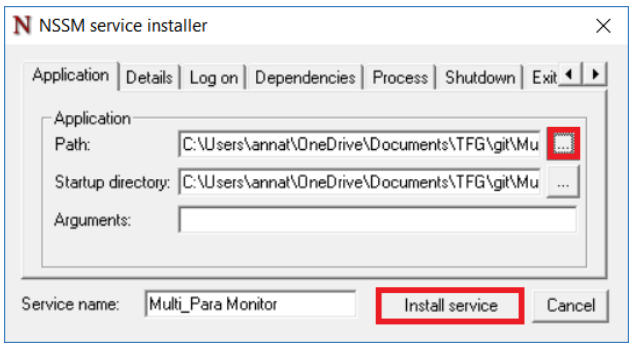
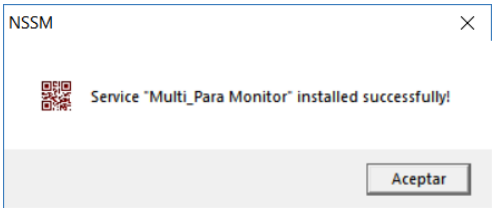


Figure A.1: Creating of a *Windows* service (1)

Thus, it is only required to select the location of the exe file of our back-end program and to click on *Install Service*. If the installation is executed successfully, a message should appear on the screen noticing the user (fig A.2). Otherwise, a message should also appear reporting a problem



(a) Selecting the desired exe file

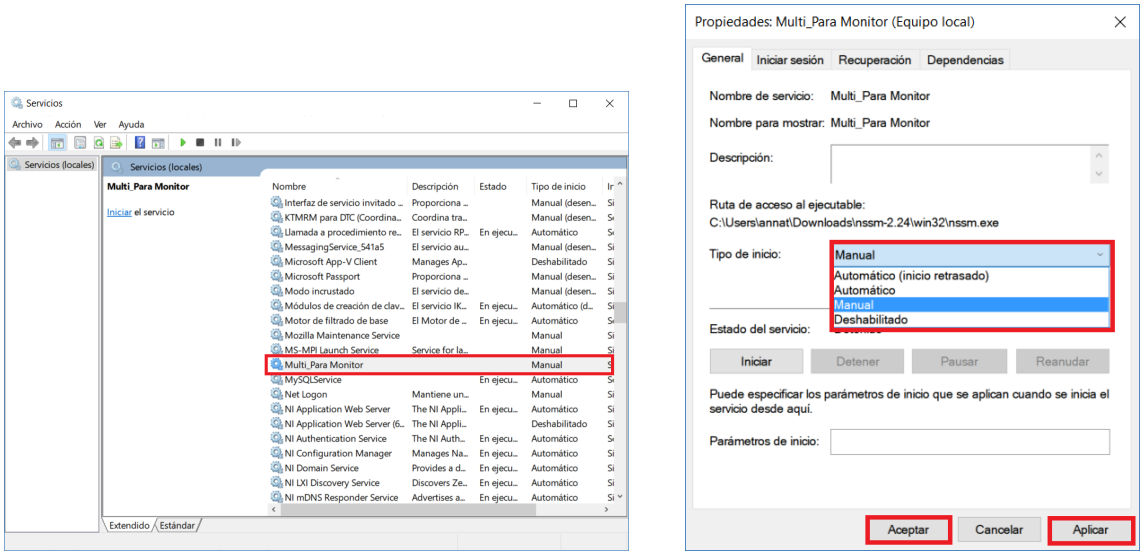


(b) After successful service creation

Figure A.2: Creating a *Windows* service (2)

A.2.3. *Windows* service setup

Finally, to check if everything was executed as expected, it is highly suggested to go to *Control Panel* → *System and Security* → *Administrative Tools* → *Services*. Once into *Services* window, look for the installed service, place the mouse over it and click on the right button to open the options menu. Subsequently, select *Properties*, then change the *startup type* from *automatic* to *manual* and apply changes (fig A.3).



(a) *Services* window

(b) Options menu of a *Windows* service

Figure A.3: Setting the service up

A.3. Building a simple *MySQL* database

As explained before, a *MySQL* database is required in order to have the already described multiparametric system working properly. Thus, a preview installation of some *MySQL* products is necessary so to be able to collect and retrieve data as well as to handle them.

A.3.1. Download of *MySQL Installer 8.0.15*

For this specific system it is needed: *MySQL Server 8.0*, *MySQL Workbench 8.0 CE*, *MySQL Router 8.0* and *MySQL Shell 8.0*.

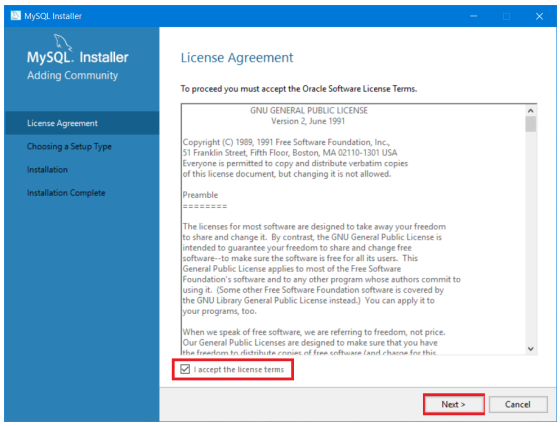
However, there isn't any necessity of installing all these products one by one. That is why, it is highly recommended to go to `dev.mysql.com/downloads/installer/` and download *MySQL Installer 8.0.15* for Windows (x86). Note that there are usually two options that can be downloaded. Hence, it is really important to install the **MSI file whose name does not contain the word *web*** and therefore it is the file weighing the most.

Moreover, it should also be taken into consideration that the *MySQL* packages used correspond to the community version and, consequently, their use is completely free of charge.

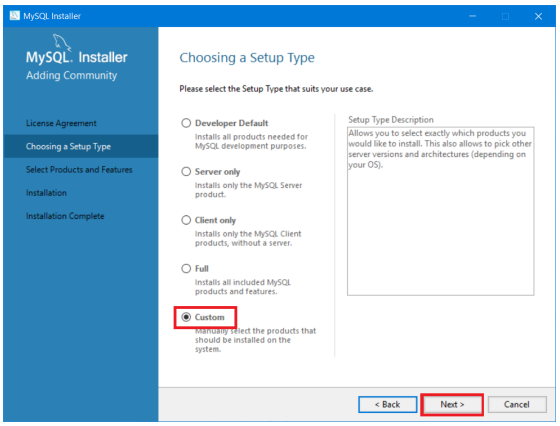
Once the file called *mysql-installer-community-8.0.15.0.msi* is downloaded, open it to proceed with the installation.

A.3.2. Installation

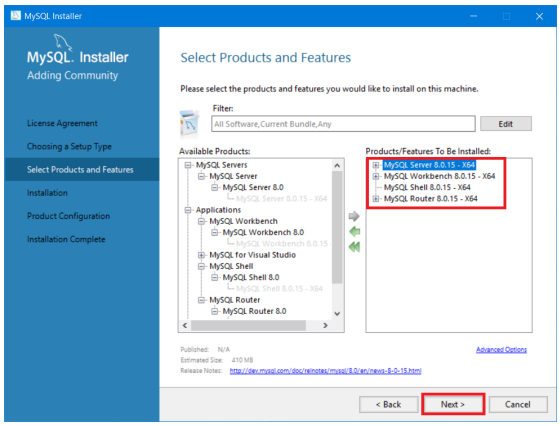
The installation consists of many little steps graphically explained below (see Fig. A.4). Here, it has been assumed that the user already installed *MYSQL* by following the instructions stated in the previous section. The read squares of figure A.4 identify the most important parts of each part, including buttons or items that must be selected by the user.



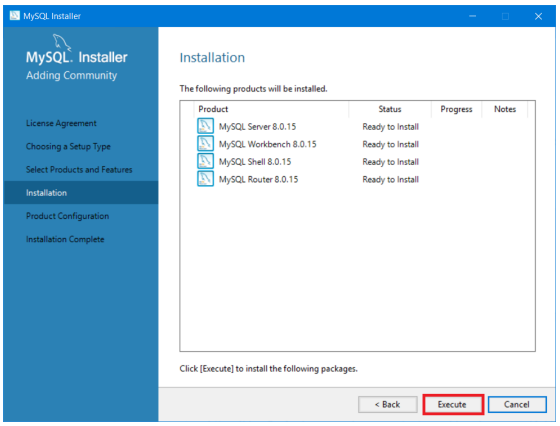
(a) Step 1. License agreement



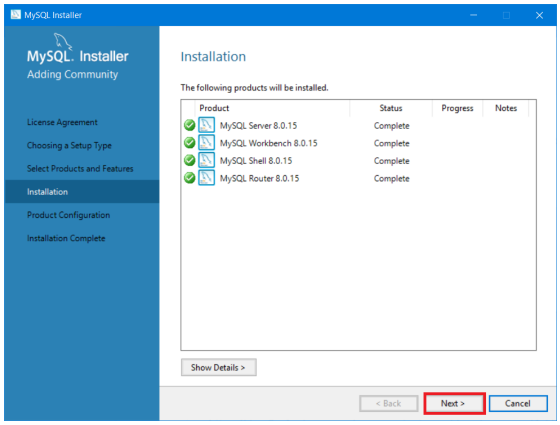
(b) Step 2. Chosing a setup type



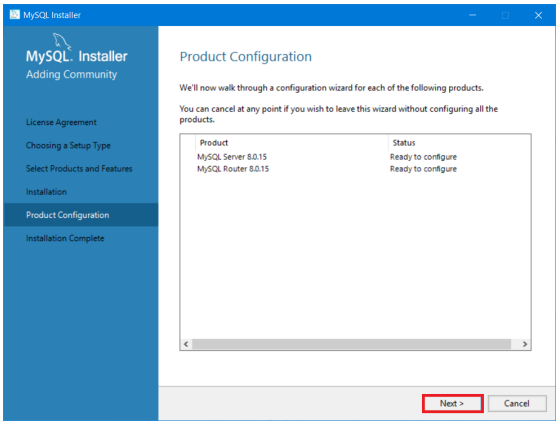
(c) Step 3. Selecting the needed products



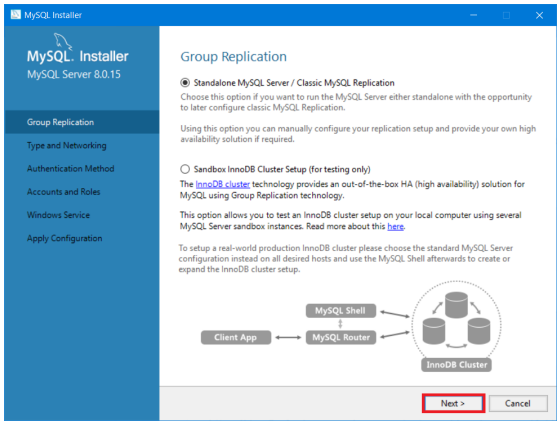
(d) Step 4. Installation overview



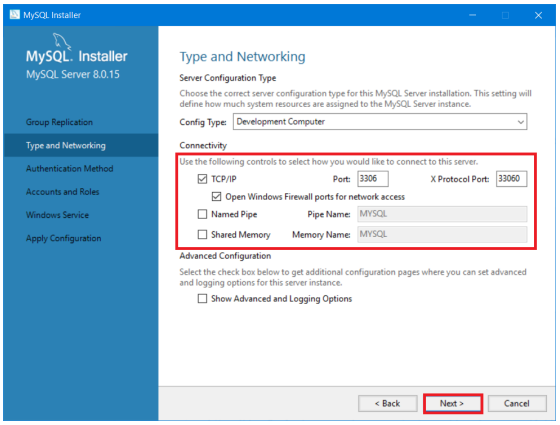
(e) Step 5. Installation execution



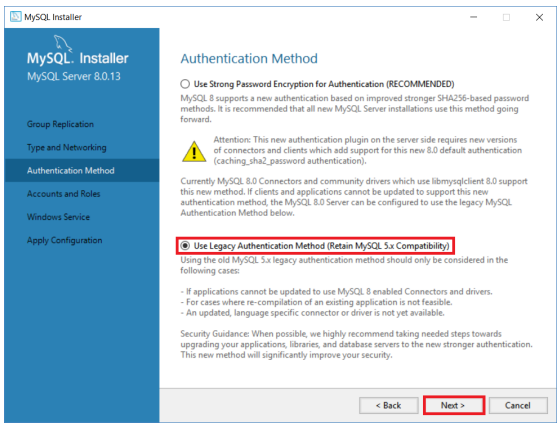
(f) Step 6. Product configuration



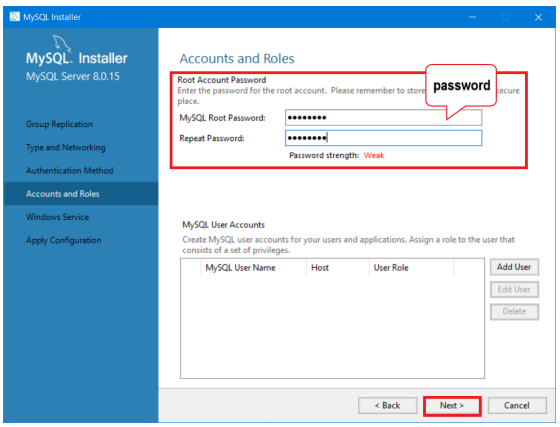
(g) Step 7. Group replication



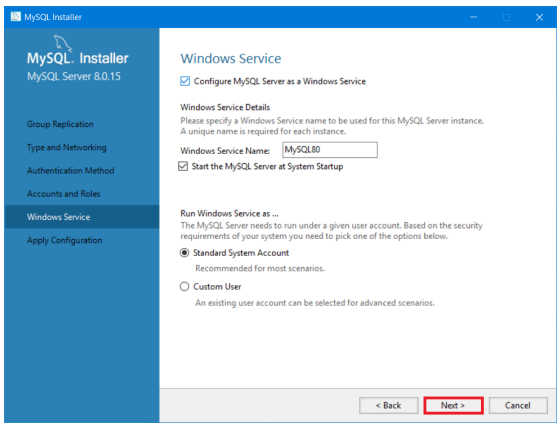
(h) Step 8. Type and networking



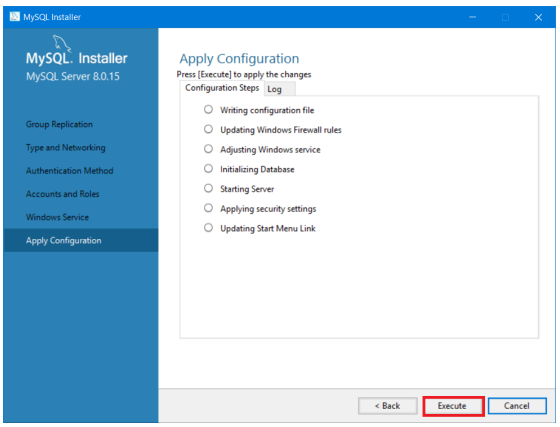
(i) Step 9. Authentication method



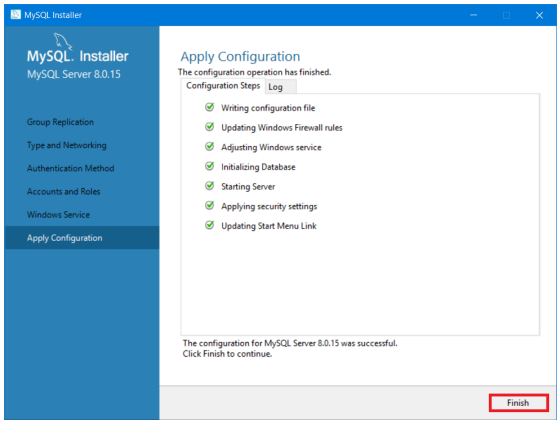
(j) Step 10. Accounts and role



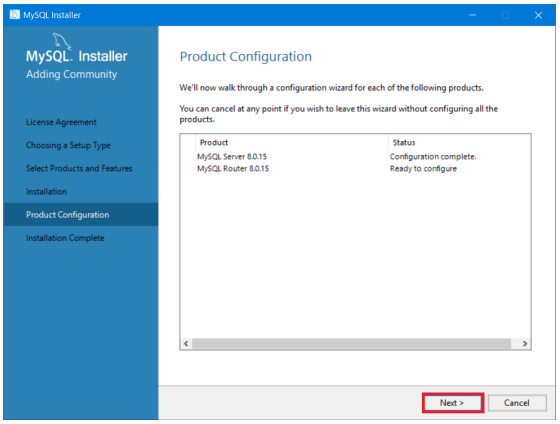
(k) Step 11. Windows service



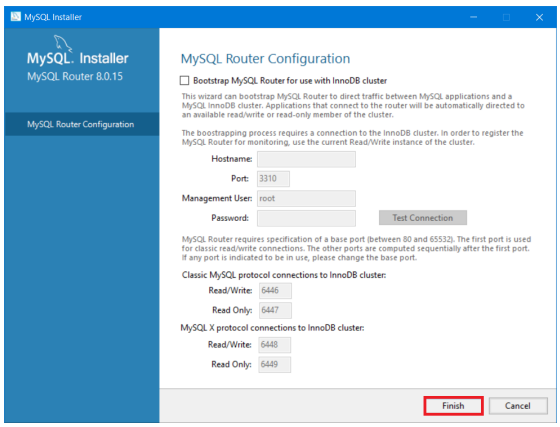
(l) Step 12. Applied configuration overview



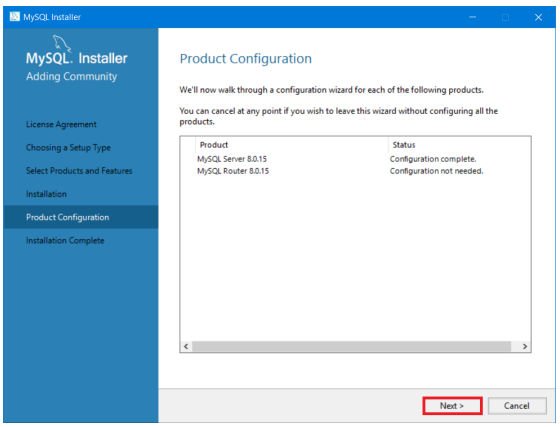
(m) Step 13. Applied configuration execution



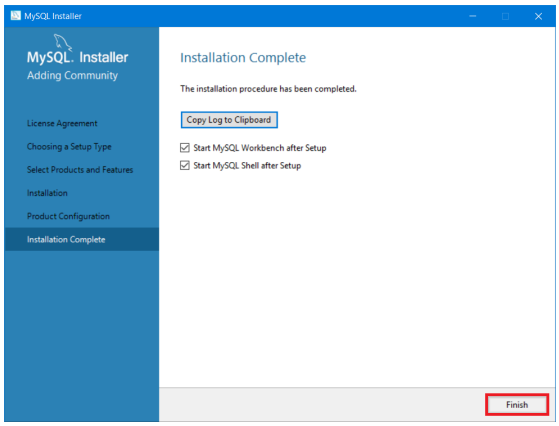
(n) Step 14. Product configuration



(o) Step 15. *MySQL Router* configuration



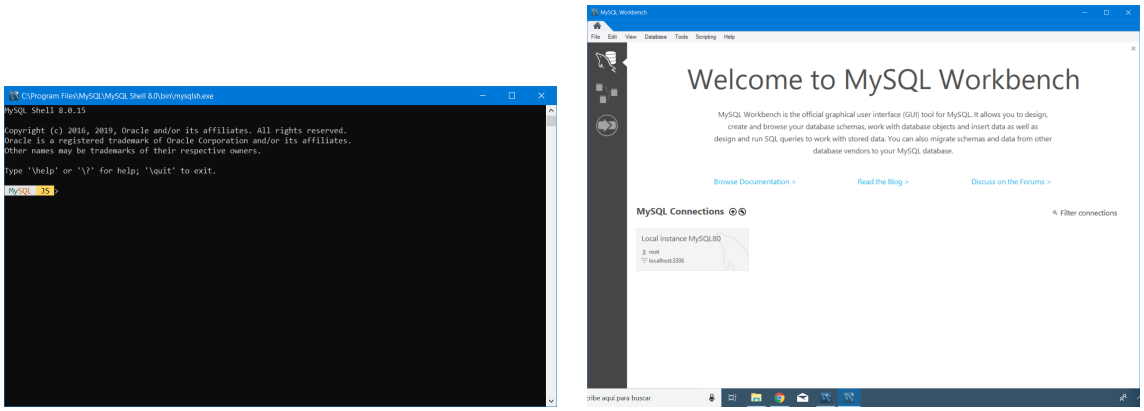
(p) Step 16. Product configuration



(q) Step 17. Installation complete

Figure A.4: Steps to follow so to correctly install *MySQL* packages

If the installation has been completed successfully, *MySQL Shell* and *MySQL Workbench* should appear automatically on the screen such as in the next couple of figures A.5 .



(a) *MySQL Shell* (b) *MySQL Workbench*

Figure A.5: Programs opened after a successfull installation of *MySQL* database

A.3.3. Restoring a previous backup

Up to now, all the needed *MySQL* packages have been installed. However, no schema with its corresponding tables has been created yet according to which data are collected and retrieved. Instead of creating a new one starting from scratch, the most recommended solution is to restore a previous backup using *MySQL Workbench* program.

However, firstly the password established during the installation should be introduced in order to access *MySQL Server* (see Fig. A.6).

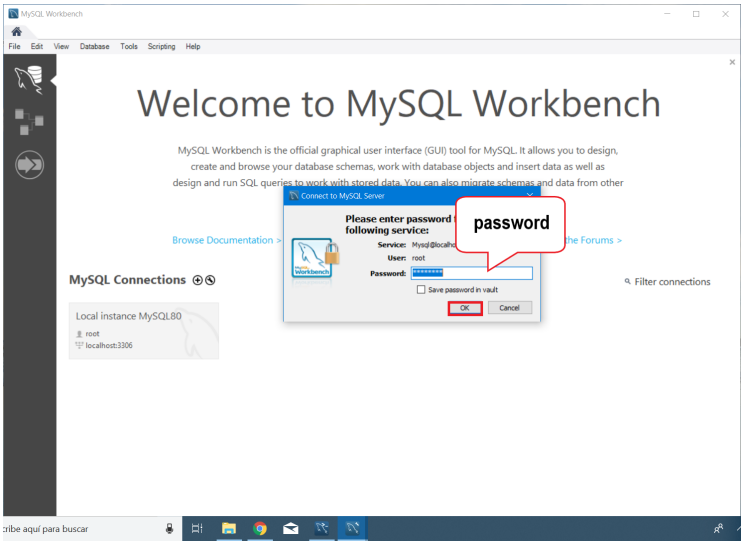


Figure A.6: Required password to access *MySQL Server*

Once the password has been correctly introduced, a new window similar to the one shown in figure A.7 should appear on the screen.

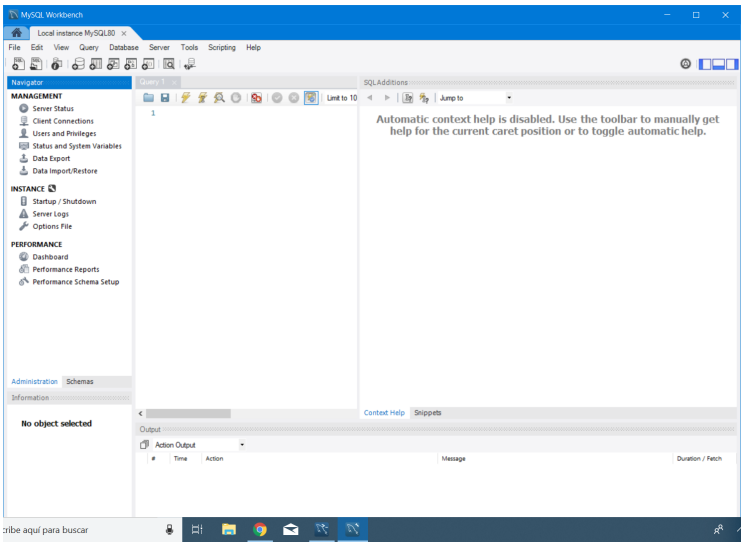


Figure A.7: *MySQL Workbench* start screen

Thereafter, it is possible to go ahead with the backup restore. To accomplish that, the five steps shown in Figure A.8 should be executed in the specified order. First, by clicking on *Data Import/Restore*, a new tab should appear allowing you to start the restore. Then, once in this tab, you should click on *Import From Self-Contained File* and therefore, look for the location of the backup file whose extension is *.sql*. For this project, the name of the backup file corresponds to *BACKUP_Initial_2.sql*. Immediately after the loading of this file, you should give a name to the default schema imported. Finally, you only have to click on *Start Import*.

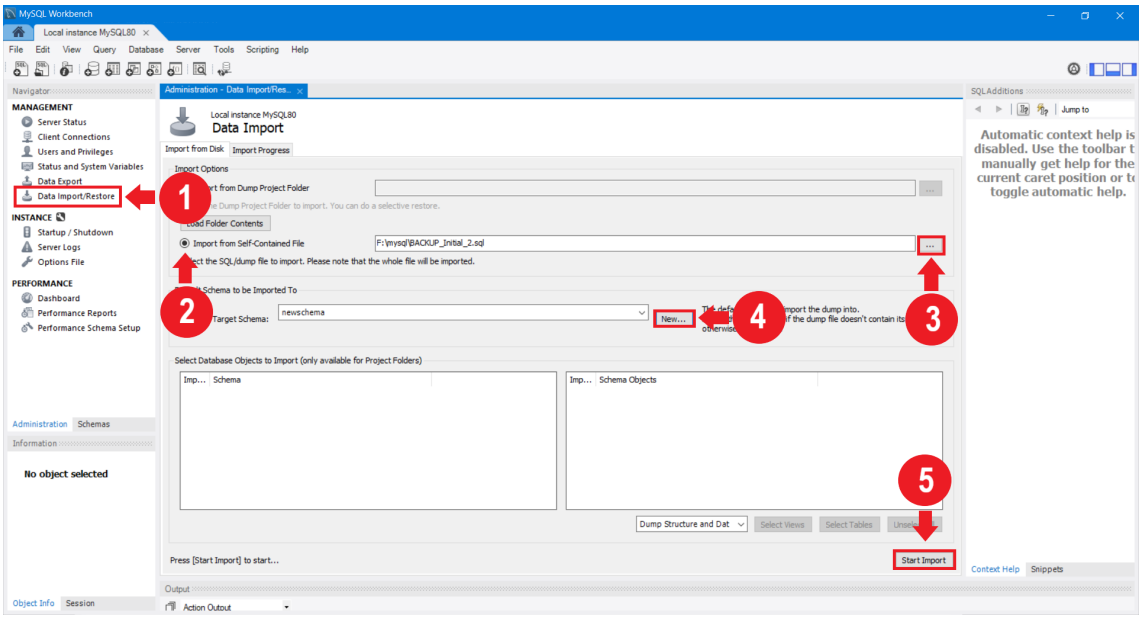


Figure A.8: Steps of the backup restore

If the restore have been carried out successfully, the opened tab should look like Figure A.9

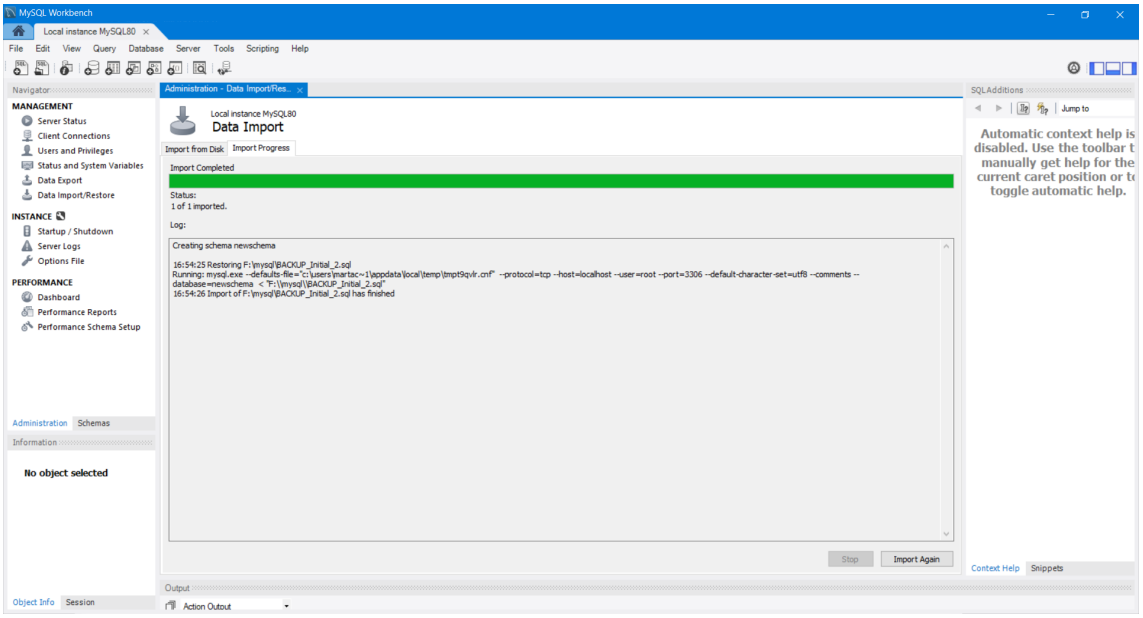


Figure A.9: Successful completion of the backup restore

To check that the schema has the expected shape, you should click on *Schemas* placed on the bottom-left side and then you should click on the refresh icon located at the top-left side. Thereafter, a schema with the name you have introduced before, should appear. To see the content of the schema, click on the plus sign at the left of the schema name (see fig A.10).

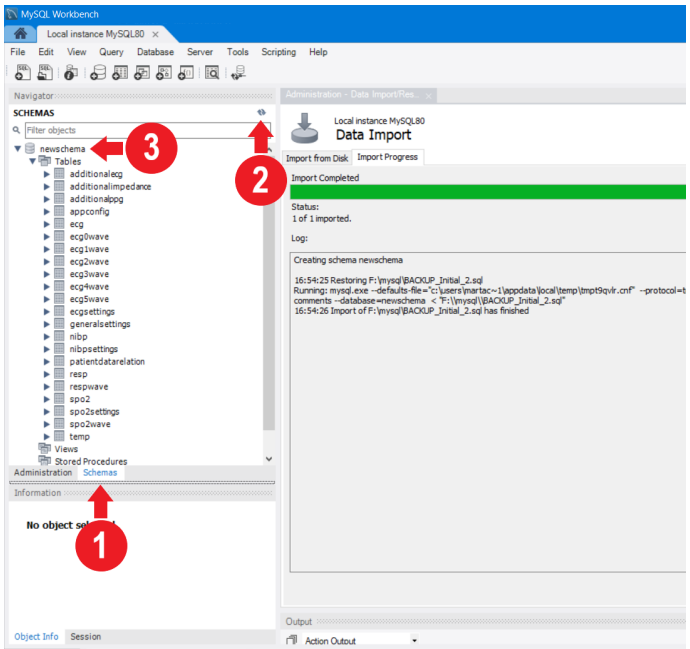


Figure A.10: Accessing to the restored schema

Finally, to verify that the tables contained in the schema are the expected ones, you should click on whatever table and click the right button of the mouse, which will show you an options menu. From this menu, click on *Select Rows - Limit 1000*. All the tables should appear empty except the one called *generalsettings* (see Fig. A.11).

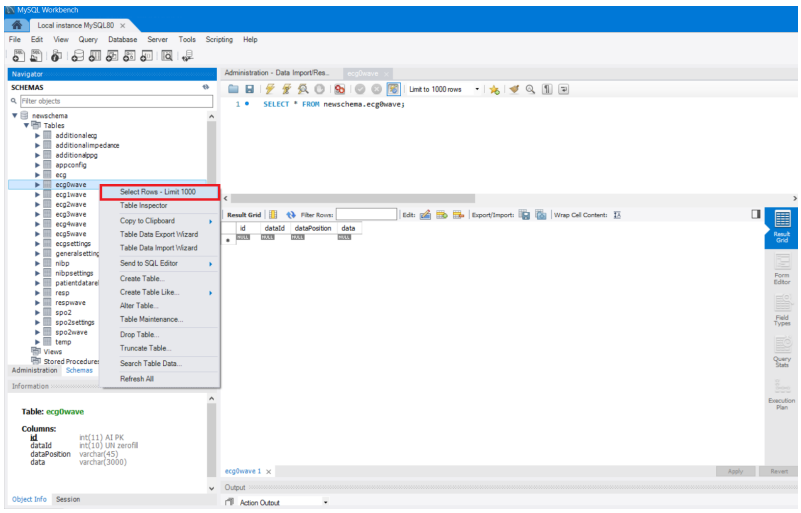


Figure A.11: Checking the content of a table of the restored schema

A.3.4. Setting a new IP to the *MySQL* database

If it were necessary to access externally to the *MySQL* database which is running in PC *A* from PC *B*, it would be possible to change the default IP (which is *localhost*) by another valid IP. Analogously, the username and the password can be also changed in this same tab.

To do so, you should come back to the welcome page of *MySQL Workbench*, place the mouse over the *MySQL* connection and then press the right button of the mouse which will show you an options menu. From this menu, click on *Edit Connection...* (see Fig. A.12).

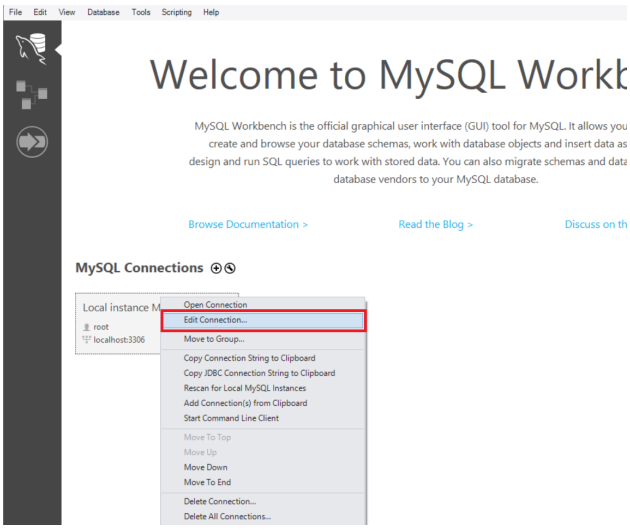


Figure A.12: Accessing to the edition of the database connection

Subsequently, a tab should appear with some options of the connection as shown in Figure A.13. As it can be seen, by changing the value of hostname, any IP can be set.

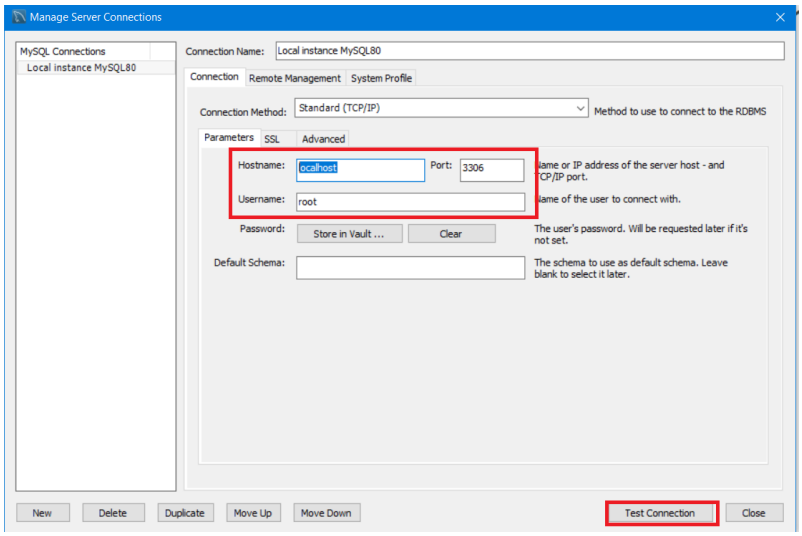


Figure A.13: Editing the IP of the database connection

APPENDIX B. STEPS TO PROPERLY START USING A MULTIPARAMETRIC SYSTEM

Once all the software parts have been successfully installed and you can make use of a hardware equipment, you are ready to start using the multiparametric system.

B.1. Checking the hardware is working well

To find out if the hardware equipment is working as expected, the executable file called *MMP-Eval.exe* must be run and a new window should appear on the screen (see Fig. B.1). What proves the system is well plugged to the PC is that, on the previous window, there are going to be two values for temperature, a value of 0 for respiration and constant lines moving from left to right for the graphs.

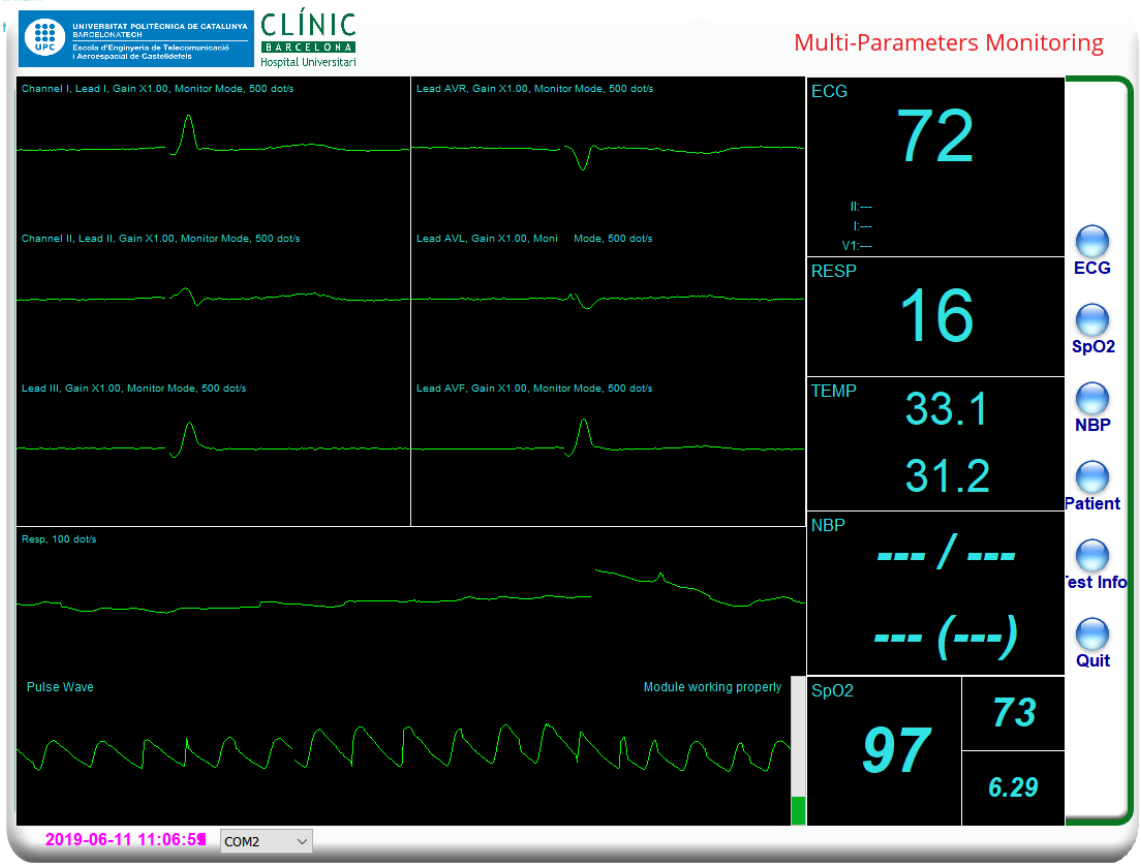


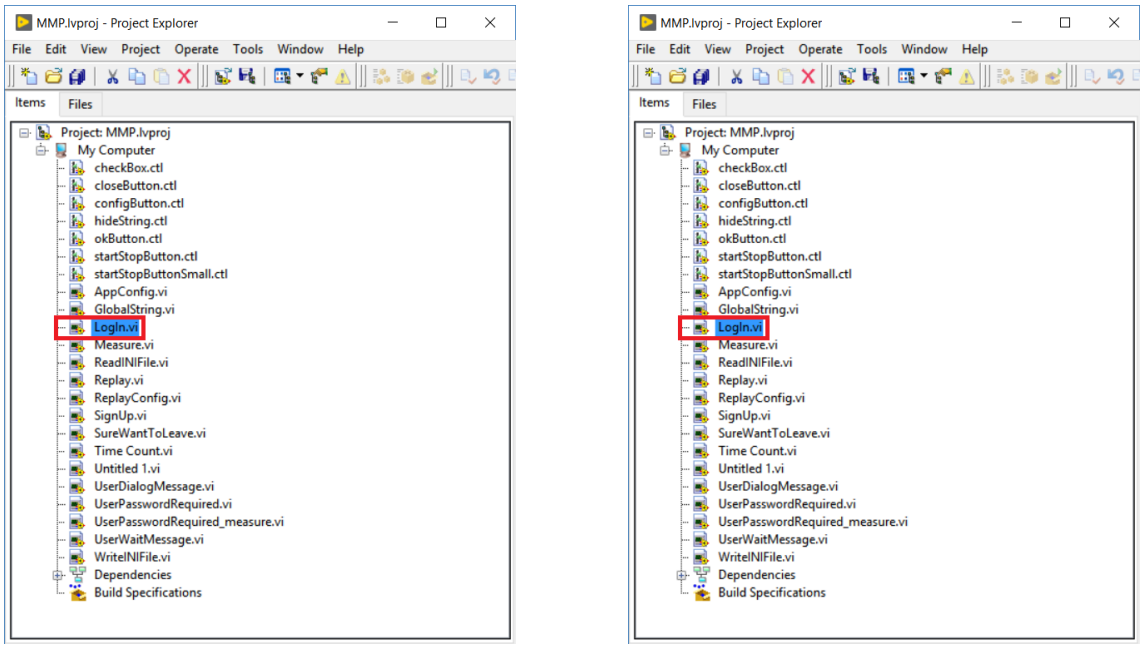
Figure B.1: Expected view of *MMP-Eval* program

In case this behavior is not noticed, check that the COM port of the hardware equipment shown in *Windows Device Manager* matches the one displayed on the previous window. Alternatively, try plugging the USB in another USB port.

B.2. Starting the system up

The multiparametric system should be started from the front-end program named *MMP_LabView*, which is the one build with *LabVIEW* software. There are two ways of starting up the system: either on debug mode or on release mode. **In any case, the user must open *LabVIEW* with administrator rights.**

On debug mode, the *LabVIEW* Project have to be opened and *LogIn.vi* have to be run. While on release mode, it is only necessary to run the executable file called *MMP_LabView.exe* (see Fig. B.2)



(a) File to be run on debug mode

(b) File to be run on release mode

Figure B.2: Files starting the system up

APPENDIX C. STEPS TO PROPERLY START USING THE MOSQUITO PREVENTION SYSTEM

Taking into account that the designed mosquito prevention system is based on automated Python scripts, a Bluetooth communication and *Arduino* scripts, Section C.1., C.2., C.3. and C.4. are intended to explain which are the software requirements and how to set up a PC to start using this system.

C.1. Installing Python 3, Anaconda and related packages

As you might know, for the proper running of the mosquito prevention system, two different programs are needed: *Python 3* which is the programming language in which the used scripts are written and *Anaconda* which is a Data Science platform making easier to perform *Python* data science and machine learning.

To get the system working as expected in our own PC, first, *Python 3* has to be downloaded from `www.python.org/downloads/` and installed, and afterward, *Anaconda* has to be downloaded from `www.anaconda.com/distribution/#download-section` and installed. For *Python* download select version 3.7.3 and for *Anaconda* download select the one corresponding to *Windows* OS for 64 bits and *Python* 3.7. When installing the downloaded executable files, accept all the legal conditions and do not change any default installation option.

Once both software installed, it is only about downloading the corresponding packages in order to be able to run the scripts the system uses. To achieve it, look for "Anaconda Prompt" in the PC search bar and select it. Finally, copy and paste each of the following lines to "Anaconda Prompt".

Notice that for each of these lines, a *Python* package should be installed. If it is not the case, something wrong happened during the installation of these packages.

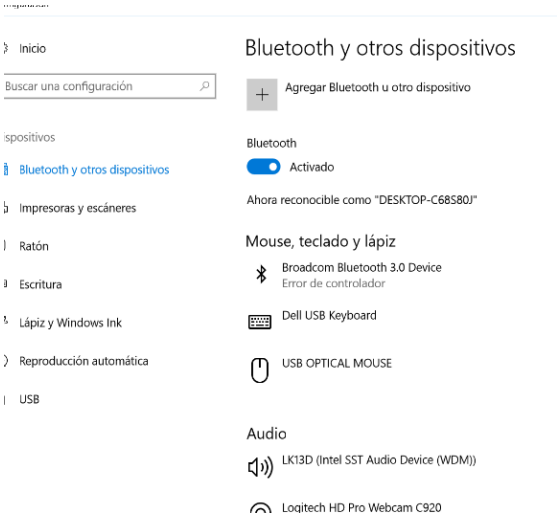
```
pip install pynmea2
pip install serial
pip install pandas
pip install pillow
pip install opencv-python
pip install tqdm
pip install argparse
conda install tensorflow
pip install pykml
pip install urlopen
```

C.2. Pairing RN-41 Bluetooth module with a PC

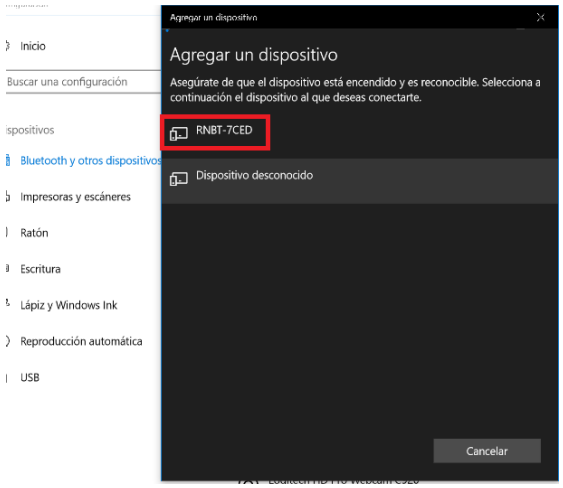
To pair RN-41 Bluetooth module with a *Windows* PC, only 4 steps should be executed. But, first of all, before starting pairing them, RN-41 Bluetooth module and the used PC should be close enough and RN-41 should be powered. If RN-41 is correctly powered and not connected to any device, a yellow led will immediately turn on and start blinking.

Once close to each other, look for ”Manage Bluetooth devices” in the PC search bar. Immediately after, a new window should appear allowing to turn on Bluetooth. Then, in the same window, click on ”Add Bluetooth or other device” (see Figure C.1(a)), which should make pop up a new floating gray window. After some seconds or even before, all the detected Bluetooth devices should appear listed in this new window (see Figure C.1(b)). From this list, select RNBT-7CED device corresponding to RN-41 module and, only then, the appearance of the current window should change to the one of Figure C.1(c). Finally, in the same window, click on ”Connect” and if the pairing has been successfully done, the window should look like the one of Figure C.1(d).

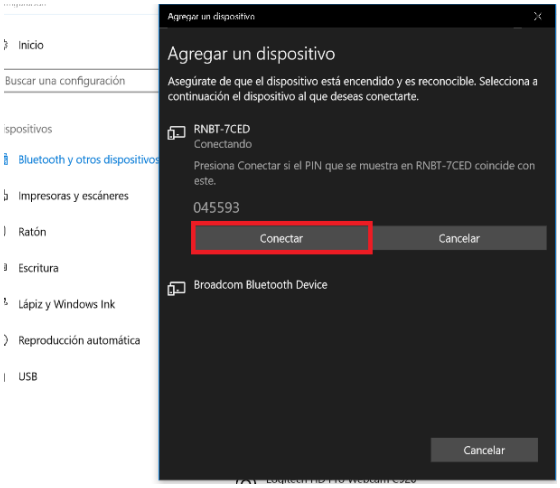
It should be noted that during this pairing process the RN-41 yellow led will remain blinking and only when there is a connection with data transmission, the blinking will stop.



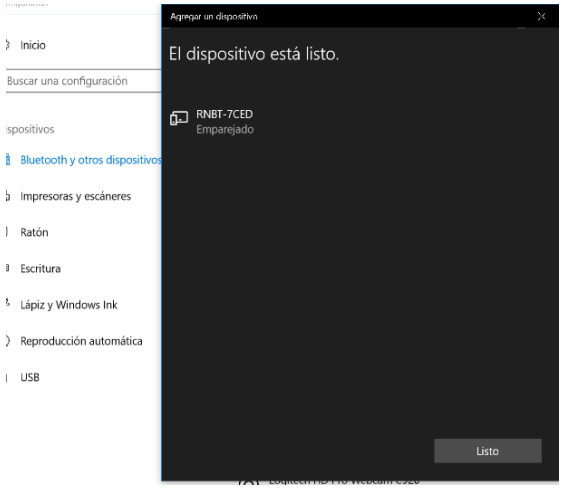
(a) Step 1. Activate Bluetooth



(b) Step 2. Search for near Bluetooth devices



(c) Step 3. Select RNBT-7CED device



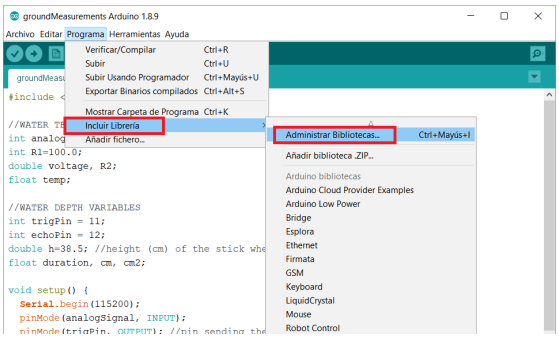
(d) Step 4. Successfully paired

C.3. Installing Arduino and related libraries

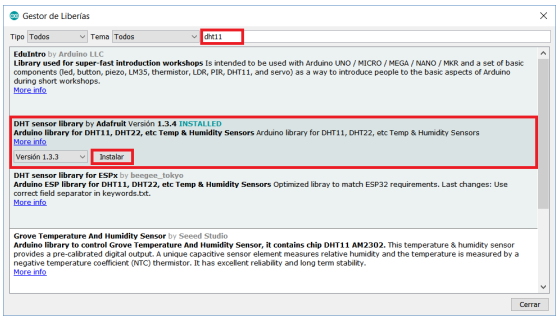
To be able to upload *Arduino* scripts to *Arduino* board, *Arduino* software should be downloaded from https://www.arduino.cc/en/Main/Donate. When installing it, keep all

the default options suggested by the installer.

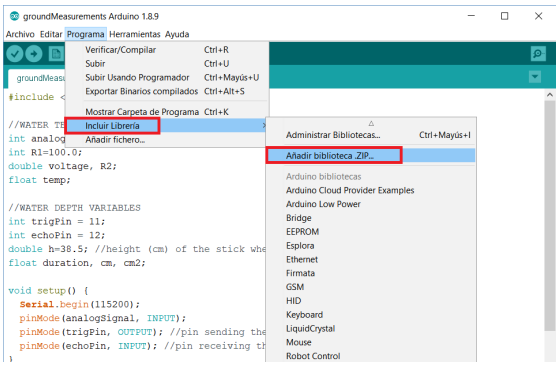
Therefore, once *Arduino* software installed, the library ”DHT.h” should be downloaded and installed in order that, from any script, DHT11 sensor data can be read. In this way, first, the library should be downloaded from https://github.com/adafruit/DHT-sensor-library.git. Afterward, the graphical steps indicated by Figure C.3. should be followed in order to directly download the library from *Arduino* libraries and, then, in order to load the library previously downloaded from *GitHub*, to *Arduino* Software. If both libraries have been successfully installed, the Arduino code named ”airMeasurements.ino” should be compiled successfully.



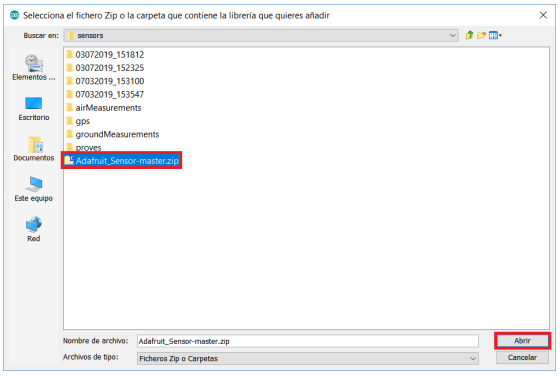
(e) Step 1. Search for *Arduino* libraries



(f) Step 2. Search for DHT11 library



(g) Step 3. Search for ZIP libraries



(h) Step 4. Select zipped library to load

C.4. Starting capturing data automatically when PC turns on

In order to have a mosquito prevention system working autonomously, the used scripts have to be run automatically.

In this way, when *LattePanda* board turns on, the three batch files C.1, C.2 and C.3 are executed. This has simply been done by saving these batch files in the *LattePanda* path called ”C:/ProgramData/Microsoft/Windows/Start Menu/Programs/StartUp”.

Notice that this path is valid for all PCs having Windows 10 OS. Moreover, due to the fact that batch files C.1 and C.2 use the same COM ports, for this first prototyped system, it is not possible to run them at the same time.

```
1 cd C:\Users\lattepanda\Desktop\AnnaSoftware\tensorflow_puddle
```

```
C:\Users\lattepanda\Anaconda3\python.exe process_puddle_full_gps.py C:/Users/
    lattepanda/Desktop/AnnaSoftware/CapturedData/ COM7 COM6 4
pause
```

Listing C.1: ”takePuddlePictures.bat”

```
cd C:\Users\lattepanda\Desktop\AnnaSoftware\tensorflow_mosquitoes
C:\Users\lattepanda\Anaconda3\python.exe process_mosquitoes_full_gps.py 64 C:/
    Users/lattepanda/Desktop/AnnaSoftware/CapturedData/ COM7 COM6 4
pause
```

Listing C.2: ”takeMosquitoesPictures.bat”

```
cd C:\Users\lattepanda\Desktop\AnnaSoftware\sensors\groundMeasurements
C:\Users\lattepanda\Anaconda3\python.exe saveData.py COM8
pause
```

Listing C.3: ”getGroundData.bat”

APPENDIX D. ”THEORETICAL_PROCESS.PY” CODE

```
import os
import sys
import argparse
import subprocess
import shlex
from tqdm import tqdm
import pandas as pd

__author__  = "ARC"
__version__ = "1.0"

def parseArguments():
  parser = argparse.ArgumentParser() # create argument parser

  # positional mandatory arguments
  parser.add_argument("script", help="path to the classify.py script", type=str
    )

  # positional optional arguments
  parser.add_argument("-o",  "--output", help="output directory", type=str,
    default="output")
  parser.add_argument("-i",  "--input", help="pictures directory", type=str,
    default="input")

  args = parser.parse_args() # parse arguments
  return args

def makedir(directory):
  if not os.path.exists(directory): os.makedirs(directory)

def execute(call):
  Process = subprocess.Popen(shlex.split(call), stdout=subprocess.PIPE, stderr=
    subprocess.PIPE)
  stdout, stderr = Process.communicate()  # blocks until execution is done
  Process.wait()
  returncode = Process.returncode
  return returncode, stdout, stderr


args   = parseArguments()
script = args.script
indir  = args.input
outdir = args.output

if not os.path.exists(indir): # check input directory
  print("Error: directory " + indir + " does not exist.")
  sys.exit(1)

if not os.path.exists(script): # check script
  print("Error: script " + script + " does not exist.")
  sys.exit(1)

try: # create output directory
```

```
  makedir(outdir)
except:
  print("Error: creating " + outdir + " directory.")
  sys.exit(1)

files = os.listdir(indir)
N = len(files)
with tqdm(total=N, unit="efd dates") as pbar:
  results = []
  classification=[]
  for f in files:
    pbar.update(1)
    call = " ".join(["python", script, indir + "/" + f]) # generate call string to classify.py
    pbar.set_description("Executing " + call)
    returncode, stdout, stderr = execute(call) # execute call
    if returncode:
      print("Error: something wrong happened while processing " + f + ".")
      continue
    else:
      print(stdout)
      print(stdout.splitlines());
      for i, line in enumerate(stdout.splitlines()):
        first, second = line.decode('ASCII').split(" (score = ")
        second = second[:-1]
        classification += [{'file': f, 'class': first, 'probability':float(second)}] # gather classify.py output
        print(second)

        if(i==0):
          results+=[{'file': f, 'type': first}] # classification result = class with highest probability
      pass

df = pd.DataFrame(classification).set_index(['file', 'class']) # save classification data in csv
df.to_csv(outdir + "/classification.csv", sep=",")
print(df)

df2 = pd.DataFrame(results).set_index(['file', 'type']) # save results data in csv
df2.to_csv(outdir + "/results.csv", sep=",")
print(df2)

sys.exit(0)
```

APPENDIX E. ”PUDDLE_PROCESS.PY” CODE

```
import warnings
from PIL import Image
import os
import sys
import argparse
import subprocess
import shlex
from tqdm import tqdm
import pandas as pd
from os import walk
import cv2
import msvcrt
import time
import serial
import pynmea2
from datetime import datetime

__author__  = "ARC"
__version__ = "1.0"

warnings.filterwarnings("ignore")

def parseArguments():
    parser = argparse.ArgumentParser() #create argument parser

    #positional mandatory arguments
    parser.add_argument("folder_path", help="folder path where pictures taken are going to be saved", type=str)
    parser.add_argument("gps_com", help="COM of GPS USB", type=str)
    parser.add_argument("environ_com", help="COM of temp and humidity sensor (Arduino Leonardo)", type=str)
    parser.add_argument("max_time", help="how much time (minutes) it is desired to run the script", type=str)
    args = parser.parse_args() #parse arguments
    return args

def get_file_paths(folderpath):
    files = next(os.walk(folderpath))[2]
    return files

def makedir(directory):
    if not os.path.exists(directory): os.makedirs(directory)

def to_jpeg(filename, folderpath):
    im=Image.open(folderpath+'/'+filename)
    filename_only=filename.split(".")
    im.save(folderpath+'/'+filename_only[0]+'.jpeg', 'JPEG')

def execute(call):
    Process = subprocess.Popen(shlex.split(call), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    stdout, stderr = Process.communicate()  # blocks until execution is done
    Process.wait()
    returncode = Process.returncode
    return returncode, stdout, stderr
```

```
52
53 def createDirectory(folder_path):
54     if not os.path.exists(folder_path): # check input directory
55         try:
56             makedir(folder_path) # create output directory
57         except:
58             print("Error: creating " + folder_path + " directory.")
59             sys.exit(1)
60
61 def checkScript(script):
62     if not os.path.exists(script): # check script
63         print("Error: script " + script + " does not exist.")
64         sys.exit(1)
65
66 def portIsUsable(portName):
67     try:
68        serial.Serial(portName)
69        return True
70     except:
71        return False
72
73
74 start_time = time.time()  # remember when the script started
75 args=parseArguments()
76 folder_path=args.folder_path
77 gps_com=args.gps_com
78 environ_com=args.environ_com
79 max_time=args.max_time
80 script="classify.py"
81 script2="draw_sizepin_from_file.py"
82 gps_filename= "gpsData.txt"
83 environ_file = "airMeasurements.csv"
84 classification_file="classification.csv"
85 results_file="results.csv"
86 kml_file="puddle_location.kml"
87 results = []
88 classification=[]
89 environ=[]
90 output_folder="/"+datetime.now().strftime("%d%m%Y_%H%M%S")
91
92 checkScript(script)
93 createDirectory(folder_path)
94 createDirectory(folder_path+output_folder) # create output directory
95 createDirectory(folder_path+output_folder+"/output") # create output folder
       within output directory
96
97
98 gps_file = open(folder_path+output_folder+"/output/"+gps_filename, "w+") #
       create gps file
99 ser = serial.Serial(gps_com, 57600, timeout=0) # open GPS COM port
100
101 ser2 = serial.Serial(environ_com, 9600, timeout=0) # open Arduino COM port
102
103 print("Loading camera...")
104 cap = cv2.VideoCapture(0) # open camera port
105 leido, frame = cap.read()
106 index=0
107 print("Camera loaded successfully!")
```

```
while((time.time() - start_time) < (int(max_time)*60)):
    if msvcrt.kbhit():
        if ord(msvcrt.getch()) == 32: # break with space tab
            break
    if leido == True:
        line = ser.readline()
        line=line.decode('utf-8')
        if (line.startswith('$GPGGA',0,len(line))): # get GPS coordinates
            msg = pynmea2.parse(line)
            lat="latitude: "+str(msg.latitude)+str(msg.lat_dir)
            lon="longitude: "+str(msg.longitude)+str(msg.lon_dir)
            print(lat+" "+lon)
            gps_file.write(str(msg)+"\n")
            gps_file.write(lat+"\n")
            gps_file.write(lon+"\n")
            print("Recording GPS data...")
            latitude=str(int(msg.latitude*1000000))+str(msg.lat_dir)
            longitude=str(int(msg.longitude*1000000))+str(msg.lon_dir)
            imagename=longitude+"_"+latitude
            f=imagename+".jpg"  # filename with coordinates
            cv2.imwrite(folder_path+output_folder+"/"+f, frame)  # save picture
            print(f+" taken successfully")
            index=index+1

            line2 = ser2.readline()
            if (line2!="" and len(line2)>20): # get air temperature and air
    humidity
                print("Recording temperature and humidity data...")
                humidity, temp, temp2=line2.decode('utf-8').split(", ")
                humidity=humidity[:-1]
                temp=temp[:2]
                temp2=temp2[:-4]
                dt=datetime.now().strftime("%d/%m/%Y %H:%M:%S") # get date time
                environ+=[{'picture':f, 'datetime': dt, 'latitude': latitude, '
    longitude': longitude, 'humidity': float(humidity), 'temperature': float(
    temp), 'temperature index': float(temp2)}]
                df4 = pd.DataFrame(environ).set_index(['picture', 'datetime', '
    latitude', 'longitude'])
                df4.to_csv(folder_path+output_folder+"/output/"+environ_file,
    sep=",") #save air data into csv


            call = " ".join(["python", script, folder_path+output_folder+ "/" +
     f]) # generate call string to classify.py
            returncode, stdout, stderr = execute(call) # execute call
            print(stderr)
            if returncode:
                print("Error: something wrong happened while processing " + f +
     ".")
                continue
            else:
                print(stdout)
                for i, line in enumerate(stdout.splitlines()):
                    first, second = line.decode('ASCII').split(" (score = ")
                    second = second[:-1]
                    classification += [{'file': f, 'type': first, 'probability'
    :float(second)}]  # gather classify.py output
                    print(second)
```

```
158
159                     if(i==0): # classification results
160                         if(first=='puddle' and float(second)<0.85):
161                             results+=[{'file': f, 'type': 'misc'}]
162                         else:
163                             results+=[{'file': f, 'type': first}]
164                     pass
165     else:
166         print("Error while accessing the camera")
167     leido, frame = cap.read()
168
169 cap.release()
170
171 df = pd.DataFrame(classification).set_index(['file']) # save classification
    data in csv
172 df.to_csv(folder_path+output_folder+"/output/"+classification_file, sep=",")
173
174 df2 = pd.DataFrame(results).set_index(['file']) # save results data in csv
175 df2.to_csv(folder_path+output_folder+"/output/"+results_file, sep=",")
176
177 for row in results: # process results to generate kml file
178     if(row['type']=='puddle'):
179         image_name=row['file'].split(".")
180         # generate call string to draw_sizepin_from_file.py
181         call = " ".join(["python",script2, folder_path+output_folder+'/'+
    image_name[0]+'.jpg',str(-1),str(-1),str(-1), '-o '+folder_path+
    output_folder+"/output/"+kml_file])
182         returncode, stdout, stderr = execute(call) # execute call
183         print(stderr)
184         if returncode:
185             print("Error: something wrong happened while calling " + script2 +
    ".")
186             continue
187         else:
188             print("Generating KML file...")
189     else:
190         image_name=row['file'].split(".")
191         # generate call string to draw_sizepin_from_file.py
192         call = " ".join(["python",script2, folder_path+output_folder+'/'+
    image_name[0]+'.jpg',str(-2),str(-2),str(-2), '-o '+folder_path+
    output_folder+"/output/"+kml_file])
193         returncode, stdout, stderr = execute(call) # execute call
194         print(stderr)
195         if returncode:
196             print("Error: something wrong happened while calling " + script2 +
    ".")
197             continue
198         else:
199             print("Generating KML file...")
200
201 sys.exit(0)
```

APPENDIX F. ”MOSQUITOES_PROCESS.PY” CODE

```
import warnings
from PIL import Image
import os
import sys
import argparse
import subprocess
import shlex
from tqdm import tqdm
import pandas as pd
from os import walk
import cv2
import msvcrt
import time
import serial
import pynmea2
from datetime import datetime

__author__  = "ARC"
__version__ = "1.0"

warnings.filterwarnings("ignore")

def parseArguments():
    parser = argparse.ArgumentParser() #create argument parser

    # Positional mandatory arguments
    parser.add_argument("width", help="width of the cropped picture", type=int)
    parser.add_argument("folder_path", help="folder path where pictures taken
    are going to be saved", type=str)
    parser.add_argument("gps_com", help="COM of GPS USB", type=str)
    parser.add_argument("environ_com", help="COM of temp and humidity sensor (
    Arduino Leonardo)", type=str)
    parser.add_argument("max_time", help="how much time (minutes) it is desired
     to run the script", type=str)
    args = parser.parse_args() #parse arguments
    return args

def get_file_paths(folderpath):
    files = next(os.walk(folderpath))[2]
    return files

def makedir(directory):
    if not os.path.exists(directory): os.makedirs(directory)


def to_jpeg(filename, folderpath):
    im=Image.open(folderpath+'/'+filename)
    filename_only=filename.split(".")
    im.save(folderpath+'/'+filename_only[0]+'.jpeg', 'JPEG')

def crop(infile,height,width):
    im = Image.open(infile)
    imgwidth, imgheight = im.size
```

```
51      for i in range(imgheight//height):
52          for j in range(imgwidth//width):
53              box = (j*width, i*height, (j+1)*width, (i+1)*height)
54              yield im.crop(box)
55
56  def execute(call):
57      Process = subprocess.Popen(shlex.split(call), stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
58      stdout, stderr = Process.communicate()  # blocks until execution is done
59      Process.wait()
60      returncode = Process.returncode
61      return returncode, stdout, stderr
62
63  def createDirectory(folder_path):
64      if not os.path.exists(folder_path): #check input directory
65          try:
66              makedir(folder_path) #create output directory
67          except:
68              print("Error: creating " + folder_path + " directory.")
69              sys.exit(1)
70
71  def checkScript(script):
72      if not os.path.exists(script): #check script
73          print("Error: script " + script + " does not exist.")
74          sys.exit(1)
75
76  def portIsUsable(portName):
77      try:
78         ser = serial.Serial(portName)
79         return True
80      except:
81         return False
82
83  start_time = time.time()  # remember when we started
84  args=parseArguments()
85  width=args.width
86  height=width
87  folder_path=args.folder_path
88  gps_com=args.gps_com
89  environ_com=args.environ_com
90  max_time=args.max_time
91  script="classify.py"
92  script2="draw_sizepin_from_file.py"
93  gps_filename= "gpsData.txt"
94  environ_file = "airMeasurements.csv"
95  classification_file="classification.csv"
96  results_file="results.csv"
97  kml_file="mosquitoes_location.kml"
98  pic_classification_file="pictures_classification.csv"
99  environ=[]
100 output_folder="/"+datetime.now().strftime("%m%d%Y_%H%M%S")
101
102 createDirectory(folder_path)
103 createDirectory(folder_path+output_folder) # create output directory
104 createDirectory(folder_path+output_folder+"/output") # create output folder
        within output directory
105 createDirectory(folder_path+output_folder+'/temp') # create new folder within
        output directory to save cropped images in it
```

```
checkScript(script)

gps_file = open(folder_path+output_folder+"/output/"+gps_filename, "w+") # create gps file
ser = serial.Serial(gps_com, 57600, timeout=0) # open GPS COM port

ser2 = serial.Serial(environ_com, 9600, timeout=0) # open Arduino COM port

print("Loading camera...")
cap = cv2.VideoCapture(0) # open camera port
leido, frame = cap.read()
index=0

while((time.time() - start_time) < (int(max_time)*60)):
    if msvcrt.kbhit():
        if ord(msvcrt.getch()) == 32: # break with space tab
            break
    if leido == True:
        line = ser.readline()
        line=line.decode('utf-8')
        if (line.startswith('$GPGGA',0,len(line))): # get GPS coordinates
            msg = pynmea2.parse(line)
            lat="latitude: "+str(msg.latitude)+str(msg.lat_dir)
            lon="longitude: "+str(msg.longitude)+str(msg.lon_dir)
            print(lat+" "+lon)
            gps_file.write(str(msg)+"\n")
            gps_file.write(lat+"\n")
            gps_file.write(lon+"\n")
            print("Recording GPS data...")
            latitude=str(int(msg.latitude*1000000))+str(msg.lat_dir)
            longitude=str(int(msg.longitude*1000000))+str(msg.lon_dir)
            imagename=longitude+"_"+latitude
            f=imagename+".jpg" # filename with coordinates
            cv2.imwrite(folder_path+output_folder + "/" +f, frame) # save picture
            print(f+" taken successfully")
            index=index+1

            line2 = ser2.readline()
            if (line2!="" and len(line2)>20): # get air temperature and air humidity
                print("Recording temperature and humidity data...")
                humidity, temp, temp2=line2.decode('utf-8').split(", ")
                humidity=humidity[:-1]
                temp=temp[:2]
                temp2=temp2[:-4]
                dt=datetime.now().strftime("%m/%d/%Y %H:%M:%S") # get date time
                environ+=[{'picture':f, 'datetime': dt, 'latitude': latitude, 'longitude': longitude, 'humidity': float(humidity), 'temperature': float(temp), 'temperature index': float(temp2)}]
                df4 = pd.DataFrame(environ).set_index(['picture', 'datetime', 'latitude', 'longitude'])
                df4.to_csv(folder_path+output_folder+"/output/"+environ_file, sep=",") #save air data into csv

    else:
        print("Error while accessing the camera")
    time.sleep(5)
```

```
    leido, frame = cap.read()

cap.release()

image_names=list()
img_paths = get_file_paths(folder_path+output_folder) #list images to crop
for index, img_path in enumerate(img_paths):
    imagename=img_path.split(".")
    print(imagename[0],imagename[1])
    image_names.append(imagename[0])
    image_to_crop=folder_path+output_folder+'/'+imagename[0]+'.jpg'

    for k,piece in enumerate(crop(image_to_crop,height,width),0): #crop image
    in jpg format
        im=Image.new('RGB', (height,width), 255)
        im.paste(piece)
        path=os.path.join(folder_path+output_folder+'/temp/'+imagename[0]+'_%s.
    jpg' % k)
        im.save(path)

files = os.listdir(folder_path+output_folder+'/temp')
N = len(files)
environ_data = pd.read_csv(folder_path+output_folder+"/output/"+environ_file,
    index_col ="picture")
results = []
classification=[]
with tqdm(total=N, unit="efd dates") as pbar: #classify images
    for f in files:
        pbar.update(1)
        call = " ".join(["python", script, folder_path+output_folder+'/temp/' +
     f]) # generate call string to classify.py
        pbar.set_description("Executing " + call)
        returncode, stdout, stderr = execute(call) # execute call
        if returncode:
            print("Error: something wrong happened while processing " + f + "."
    )
            continue
        else:
            isClassified=False
            for i, line in enumerate(stdout.splitlines()):
                first, second = line.decode('ASCII').split(" (score = ")
                second = second[:-1]
                results += [{'file': f, 'class': first, 'probability':float(
    second)}]

                org_file,remaining=f.split("_")
                org_file_first=org_file+'.jpg'
                environ=environ_data.loc[org_file_first]

                if(not isClassified): # classification results
                    if(first=="aedesalbopictus"):
                        if(environ.humidity>30 and environ.humidity<90 and
    environ.temperature>15 and environ.temperature<35):
                            classification+=[{'file': f, 'type': first, 'num'
    :1}]
                            isClassified=True
                        else:
                            isClassified=False
```

```
                    elif(first=="anophelesatroparvus"):
                        if(environ.humidity>40 and environ.humidity<90 and
    environ.temperature>10 and environ.temperature<42):
                            classification+=[{'file': f, 'type': first, 'num'
    :1}]
                            isClassified=True
                        else:
                            isClassified=False
                    elif(first=="culexpipiens"):
                        if(environ.humidity>40 and environ.humidity<90 and
    environ.temperature>15 and environ.temperature<32):
                            classification+=[{'file': f, 'type': first, 'num'
    :1}]
                            isClassified=True
                        else:
                            isClassified=False
                    else:
                        classification+=[{'file': f, 'type': first, 'num':"NaN"
    }]
                        isClassified=True
            pass

df = pd.DataFrame(results).set_index(['file', 'class']) #save classification
    data to csv
df.to_csv(folder_path+output_folder+"/output/"+classification_file, sep=",")

df2 = pd.DataFrame(classification).set_index(['file', 'type']) #save results
    data to csv
df2.to_csv(folder_path+output_folder+"/output/"+results_file, sep=",")

pictures_classification=[]
for image_name in image_names: #count mosquitoes per picture
    num_aedes=0
    num_anopheles=0
    num_culex=0
    for row in classification:
        original_image_name=row['file'].split('-')
        if((row['num']!='NaN') and (original_image_name[0]==image_name)):
            if(row['type']=='aedesalbopictus'):
                num_aedes+=1
            elif(row['type']=='anophelesatroparvus'):
                num_anopheles+=1
            elif(row['type']=='culexpipiens'):
                num_culex+=1

    pictures_classification+=[{'file': image_name, '#aedes': num_aedes, '#
    anopheles':num_anopheles, '#culex':num_culex}]
     # generate call string to draw_sizepin_from_file.py
    call = " ".join(["python",script2, folder_path+output_folder+'/'+image_name
    +'.jpg',str(num_aedes),str(num_anopheles),str(num_culex), '-o '+folder_path
    +output_folder+"/output/"+kml_file])
    returncode, stdout, stderr = execute(call) # execute call
    print (stdout)
    print (stderr)
    if returncode:
        print("Error: something wrong happened while calling " + script2 + ".")
        continue
    else:
```

```
255         print("Generating KML file...")
256
257 df3 = pd.DataFrame(pictures_classification).set_index(['file']) #save
    mosquitoes per picture
258 df3.to_csv(folder_path+output_folder+"/output/"+pic_classification_file, sep=",
    ")
259 sys.exit(0)
```

APPENDIX G. ”DRAW_SIZEPIN_FROM_FILE.PY” CODE

```
from os import path
import os
import sys
from lxml import etree
from pykml.parser import Schema
from pykml.factory import KML_ElementMaker as KML
from pykml import parser
import argparse
from math import *

__author__  = "ARC"
__version__ = "1.0"

quotient=1 # to reduce or increase pin size

def parseArguments():
    parser = argparse.ArgumentParser() #create argument parser

    # positional mandatory arguments
    parser.add_argument("file", help="picture name with extension", type=str)
    parser.add_argument("aedes_num", help="number of mosquitoe aedes per
    picture", type=int)
    parser.add_argument("anopheles_num", help="number of mosquitoe anopheles
    per picture", type=int)
    parser.add_argument("culex_num", help="number of mosquitoe culex per
    picture", type=int)
    parser.add_argument("-o", "--output", help="output directory", type=str,
    default="prova.kml")  # optional arguments
    args = parser.parse_args() #parse arguments
    return args

def get_kml_styles(aedes_num, anopheles_num, culex_num):
    styles=[]
    if(aedes_num>0):
        pin_size_aedes=log10(aedes_num/quotient)
        doc= KML.Style(
                KML.IconStyle(
                    KML.color('FFFF7800'),
                    KML.scale(pin_size_aedes),
                    KML.Icon(
                      KML.href("http://maps.google.com/mapfiles/kml/paddle/wht-
    blank.png"), #pin
                    )
                ),
                id='aedes'+str(pin_size_aedes)
            )
        styles+=doc
    if(anopheles_num>0):
        pin_size_anopheles=log10(anopheles_num/quotient)
        doc= KML.Style(
                KML.IconStyle(
                    KML.color('FF143CFF'),
                    KML.scale(pin_size_anopheles),
```

```
                    KML.Icon(
                      KML.href("http://maps.google.com/mapfiles/kml/paddle/wht-blank.png"), #pin
                    )
                ),
                id='anopheles'+str(pin_size_anopheles)
            )
        styles+=doc
    if(culex_num>0):
        pin_size_culex=log10(culex_num/quotient)
        doc=KML.Style(
                KML.IconStyle(
                    KML.color('ff14f0ff'),
                    KML.scale(pin_size_culex),
                    KML.Icon(
                      KML.href("http://maps.google.com/mapfiles/kml/paddle/wht-blank.png"), #pin
                    )
                ),
                id='culex'+str(pin_size_culex)
            )
        styles+=doc
    if(aedes_num==-1 and anopheles_num==-1 and culex_num==-1):
        doc=KML.Style(
                KML.IconStyle(
                    KML.scale(1),
                    KML.Icon(
                      KML.href("http://maps.google.com/mapfiles/kml/shapes/water.png"), #pin
                    )
                ),
                id='puddle'
            )
        styles+=doc
    if((aedes_num==-2 and anopheles_num==-2 and culex_num==-2) or (aedes_num==0 and anopheles_num==0 and culex_num==0)):
        doc=KML.Style(
                KML.IconStyle(
                    KML.scale(1),
                    KML.Icon(
                      KML.href("http://maps.google.com/mapfiles/kml/shapes/camera.png"), #pin
                    )
                ),
                id='misc'
            )
        styles+=doc

    return styles

def get_lat_lon_from_file(file):
  file_parts=file.split("/")
  filename=file_parts[len(file_parts)-1]
  filename_parts=filename.split("_")
  lon=filename_parts[0]
  lat=filename_parts[1][:-3]

  if (lon.find("W")!=-1):
```

```
    lon='-'+lon[:-1]
  else:
    lon=lon[:-1]
  lon=float(lon)/1000000

  if (lat.find("S")!=-1):
    lat='-'+lat[:-2]
  else:
    lat=lat[:-2]
  lat=float(lat)/1000000

  lat_lon_str=str(lon)+","+str(lat)
  return lat_lon_str


args=parseArguments()
file=args.file
aedes_num=args.aedes_num
anopheles_num=args.anopheles_num
culex_num=args.culex_num
kml_filename=args.output

styles=get_kml_styles(aedes_num, anopheles_num, culex_num)

kml_file = os.path.isfile(kml_filename) # check is kml file exists
if kml_file:
    kml_file = path.join(kml_filename)
    with open(kml_file) as f:
        doc = parser.parse(f).getroot() # open kml file
else:
    doc=KML.kml(KML.Folder(KML.name("CNN output"))) # create kml file

for style in styles:
        doc.Folder.append(style)

if(aedes_num>0):
    placemark = KML.Placemark(
      KML.name(""),
      KML.styleUrl("aedes"+str(log10(aedes_num/quotient))),
      KML.Point(
        KML.coordinates(get_lat_lon_from_file(file)),
      )
    )
    doc.Folder.append(placemark)
if(anopheles_num>0):
    placemark = KML.Placemark(
        KML.name(""),
        KML.styleUrl("anopheles"+str(log10(anopheles_num/quotient))),
        KML.Point(
            KML.coordinates(get_lat_lon_from_file(file)),
        )
    )
    doc.Folder.append(placemark)
if(culex_num>0):
    placemark = KML.Placemark(
        KML.name(""),
        KML.styleUrl("culex"+str(log10(culex_num/quotient))),
        KML.Point(
```

```
            KML.coordinates(get_lat_lon_from_file(file)),
        )
    )
    doc.Folder.append(placemark)
if(aedes_num==-1 and anopheles_num==-1 and culex_num==-1):
    placemark = KML.Placemark(
        KML.name(""),
        KML.styleUrl("puddle"),
        KML.Point(
            KML.coordinates(get_lat_lon_from_file(file)),
        )
    )
    doc.Folder.append(placemark)
if((aedes_num==-2 and anopheles_num==-2 and culex_num==-2) or (aedes_num==0 and
     anopheles_num==0 and culex_num==0)):
    placemark = KML.Placemark(
        KML.name(""),
        KML.styleUrl("misc"),
        KML.Point(
            KML.coordinates(get_lat_lon_from_file(file)),
        )
    )
    doc.Folder.append(placemark)

print("KML file generated!")
outfile = open(kml_filename,'w')
outfile.write(etree.tostring(doc).decode("utf-8")) # save kml file
outfile.close();
```

APPENDIX H. ”SAVE_BLUETOOTH_DATA.PY” CODE

```
import serial
from datetime import datetime
import argparse
import time
import msvcrt

__author__  = "ARC"
__version__ = "1.0"

def parseArguments(): #create argument parser

    #positional mandatory arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("com", type=str)
    args = parser.parse_args() #parse arguments
    return args

def portIsUsable(portName): #check if COM port is usable
    try:
       ser = serial.Serial(portName)
       return True
    except:
       return False

args=parseArguments()
com=args.com
write_to_file_path = "groundMeasurements.txt"
output_file = open(write_to_file_path, "w+",errors='ignore')

stop=False
while (stop==False):
    if msvcrt.kbhit():
        if ord(msvcrt.getch()) == 32:
            break

    elif(portIsUsable(com)==False):
        # if port no usable --> wait 2 seconds
        print('Waiting for bluetooth connection...')
        time.sleep(2)
    else:
        # if port usable --> record data
        i=0
        ser = serial.Serial(com, 115200,timeout=None)
        print("Recording data of ground temperature and water level...")
        line = ser.readline();
        line = line.decode("ISO-8859-1")
        if (line!=""):
            string=line
            output_file.write(datetime.now().strftime("%d/%m/%Y %H:%M:%S ")+" "
    +string)
            i=i+1
        while(i<10):  # record up to 5 samples
          line = ser.readline()
```

```
53          line = line.decode("ISO-8859-1")
54          if (line!=""):
55            string=line
56            output_file.write(datetime.now().strftime("%d/%m/%Y %H:%M:%S ")+" "
    +string)
57            i=i+1
58
59        if(i==10): # after recording 10 samples --> wait 20 seconds
60          print("Data gathered successfully!")
61          ser.flushOutput()
62          ser.close()
63            stop=True
```

APPENDIX I.
”PUDDLE_AND_BTB_DATA_COMPARISON.PY” CODE

```
import warnings
import os
import sys
import argparse
import subprocess
import shlex
import pandas as pd
from os import walk
import msvcrt
import time
from datetime import datetime, timedelta

__author__  = "ARC"
__version__ = "1.0"

warnings.filterwarnings("ignore")

def parseArguments():
    parser = argparse.ArgumentParser() #create argument parser

    #positional mandatory arguments
    parser.add_argument("folder_path", help="folder path where results are saved", type=str)
    parser.add_argument("btb_path", help="folder path where ground data is saved", type=str)
    args = parser.parse_args() #parse arguments
    return args

args=parseArguments()
folder_path=args.folder_path
btb_path=args.btb_path
btb_filename= "groundMeasurements.txt"
environ_file = "airMeasurements.csv"
res_file="results.csv"

environ_data = pd.read_csv(folder_path+'/'+environ_file)
res_data = pd.read_csv(folder_path+'/'+res_file)

i=0
while(i<len(environ_data.index)):

    if msvcrt.kbhit():
        if ord(msvcrt.getch()) == 32: # break with space tab
            break

    pic=environ_data.iloc[i].picture # get picture name
    date=environ_data.iloc[i].datetime
    datetime_pic=datetime.strptime(date, '%d/%m/%Y %H:%M:%S') # get date and time in which the picture was taken
    typee=res_data.iloc[i].type # get the label assigned to the picture after being processed
```

```
48
49      if(typee=="puddle"): # in case the label is "puddle", picture data is
        processed again
50
51          btb_file = open(btb_path+'/'+btb_filename, "r")
52          temp=0
53          depth=0
54          samples=0
55
56          for line in btb_file:
57
58              if msvcrt.kbhit():
59                  if ord(msvcrt.getch()) == 32: # break with space tab
60                      break
61              if(len(line)>10):
62                  date = line.split(" ")
63                  date_=date[0]
64                  print(date_)
65                  time_=date[1]
66                  temp_str=date[3][:-8]
67                  depth_str=date[5][:-3]
68                  datetime_environ = datetime.strptime(date_+" "+time_, '%d/%m/%Y
        %H:%M:%S')
69
70                 #assess ground environment conditions within an interval of
        +-10min from the moment the picture was taken
71                  if(datetime_pic+timedelta(minutes=10)>datetime_environ or
        datetime_pic-imedelta(minutes=10)<datetime_environ):
72                      temp=temp+float(temp_str) #add all temperatures
73                      depth=depth+float(depth_str) #add all depths
74                      samples=samples+1
75
76          average_temp=temp/samples # get temperature average of the moment the
        picture was taken
77          average_depth=depth/samples # get depth average of the moment the
        picture was taken
78
79          # if conditions are not suitables for larvae growth, label is changed
        from "pudddle" to "misc"
80          if(temp<6.2 or temp>33.2 or depth>120 or depth<3):
81              res_data.iloc[i].type="misc"
82              print(res_data)
83              res_data.to_csv(folder_path+'/'+res_file, sep=",",index=False)
84      i=i+1
85
86  sys.exit(0)
```

APPENDIX J. ”AIRMEASUREMENTS.INO” CODE

```
#include <DHT.h>
#define DHTPIN 2 //Definition of digital pin where the sensor is connected
#define DHTTYPE DHT11 //Sensor type

DHT dht(DHTPIN, DHTTYPE); //DHT11 sensor initialization

void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  //AIR TEMPERATURE AND HUMIDITY READING
  delay(5000); //5 seconds between measurements
  float h = dht.readHumidity(); //relative humidity
  float t = dht.readTemperature(); //temperature in Celsius degrees

  if (isnan(h) || isnan(t)) { //checking if any error while reading
    measurements
    Serial.println("Error while reading DHT11 sensor data");
    return;
  }

  float hic = dht.computeHeatIndex(t, h, false); //heat index in Celsius
    degrees
  Serial.print(h);
  Serial.print(" %, ");
  Serial.print(t);
  Serial.print(" degrees, ");
  Serial.print(hic);
  Serial.println(" degrees");
}
```

APPENDIX K. ”GROUNDMEASUREMENTS.INO” CODE

```
#include <math.h>

//WATER TEMPERATURE VARIABLES
int analogSignal=A0;
int R1=100.0;
double voltage, R2;
float temp;

//WATER DEPTH VARIABLES
int trigPin = 11;
int echoPin = 12;
double h=38.5; //height (cm) of the stick where the ultrasounds sensor is
    placed
float duration, cm, cm2;

void setup() {
  Serial.begin(115200);
  pinMode(analogSignal, INPUT);
  pinMode(trigPin, OUTPUT); //pin sending the transmitted pulse
  pinMode(echoPin, INPUT); //pin receiving the echo of the transmitted pulse
}

void loop() {
  //WATER TEMPERATURE READING
  voltage=5.0*analogRead(A0)/1023.0; //real voltage value
  if(voltage>0){
    R2=voltage*R1/(5.0-voltage); //PT100 resistance
    temp=(R2/100.0-1)/0.00385; //temperature from the previous resistance
    temp=1.0477*temp-14.499; //calibration
    Serial.print(temp);
    Serial.print("degrees, ");
  }

  //WATER DEPTH READING
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5); //5s to go from LOW to HIGH pulse
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); //10s to go from HIGH to LOW pulse
  digitalWrite(trigPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH); //time between the sent pulse and its echo

  cm =(duration/2)/29.41; //cm above water from the previous duration
  cm2=h-cm; //cm below water from the previous distance
  Serial.print(cm);
  Serial.print("cm, ");
  Serial.print(cm2);
  Serial.print("cm");
  Serial.println();
  delay(1000); //1 second between measurements
}
```